

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
Московский Государственный Технический Университет имени Н.Э. Баумана

**Лабораторная работа № 1.**По курсу «Операционные системы».  
Тема: «Прерывание таймера INT 08h и его функции».

Студент: Лаврова А. А.  
Группа: ИУ7-55Б.  
Преподаватель: Рязанова Н. Ю.

Макаренко В. А.

Москва, 2019.

**Тик** - период времени между двумя последующими прерываниями таймера.

**Основной тик** - период времени равный n тикам таймера (число n зависит от конкретного варианта системы).

**Квант** - период времени, на который планировщик предоставляет процессор каждому процессу системы.

**Функции системного таймера в защищенном режиме**

# **Unix**

* по тику:
  + инкремент времени использования процессора текущим процессом
  + инкремент часов и других таймеров системы
  + декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов на выполнение при достижении нулевого значения счетчика
* по кванту:
  + посылка текущему процессу сигнала SIGXCPU, если истек выделенный ему квант использования процессорного времени
* по главному тику:
  + перепланирование работы планировщика-диспетчера
  + пробуждение в нужные моменты времени системных процессов, таких, как swapper и pagedaemon (процедура wakeup перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»)
  + декремент времени, оставшегося до отправления одного из сигналов:
    - SIGALARM (декремент будильников)
    - SIGPROF (измерение времени работы процесса)
    - SIGVTALARM (измерение времени работы процесса в режиме задачи)

# **Windows**

* по тику:
  + инкремент системного времени
  + декремент счетчиков отложенных задач
  + декремент остатка кванта текущего потока
  + активизация обработчика ловушки профилирования ядра (добавление процесса в очередь DPC)
* по кванту:
  + инициация диспетчеризации потоков (посредством добавления соответствующего объекта DPC в очередь).
* по главному тику:
* инициализация диспетчера настройки баланса (путем освобождения объекта «событие», на котором он ожидает).

**Пересчет динамических приоритетов пользовательских процессов для ОС семейств Linux/Unix**

В каждой Unix-машине есть аппаратный таймер, который вырабатывает прерывание в системе через определенные промежутки времени. Период времени между двумя такими прерываниями (тиками) в ОС Unix равен 10 мсек. Обработчик прерывания по таймеру является вторым по приоритету – первым является обработчик прерывания по сбою питания.

Традиционное ядро UNIX является строго невытесняющим. Если процесс выполняется в режиме ядра (например, в течение исполнения системного вызова или прерывания), то ядро не заставит такой процесс уступить процессор какому-либо высокоприоритетному процессу. Выполняющийся процесс может только добровольно освободить процессор в случае своего блокирования в ожидании ресурса, иначе он может быть вытеснен при переходе в режим задачи. Реализация ядра невытесняющим решает множество проблем синхронизации, связанных с доступом нескольких процессов к одним и тем же структурам данных ядра.

Приоритет процесса задается любым целым числом от 0 до 127 (приоритеты от 0 до 49 – зарезервировано для ядра, а прикладные процессы обладают диапазоном от 50 до 127). Чем меньше число, тем выше приоритет процесса.

**В структуре proc содержатся поля, относящиеся к приоритетам:**

|  |  |  |
| --- | --- | --- |
| p\_pri | Текущий приоритет планирования | Используется для хранения временного приоритета для выполнения в режиме ядра |
| p\_uspri | Приоритет режима задачи | Используется для хранения приоритета, который будет назначен процессу при возврате в режим задачи |
| p\_cpu | Результат последнего измерения использования процессора | Содержит величину результата последнего сделанного измерения использования процессора процессом. Инициализируется нулем. |
| p\_nice | Фактор nice, устанавливаемый пользователем | Увеличение значения приводит к уменьшению приоритета. |

Планировщик использует *p\_pri* для принятия решения о том, какой процесс отправить на выполнение. Когда процесс находится в режиме задачи, *p\_pri* равен *p\_usrpri*.

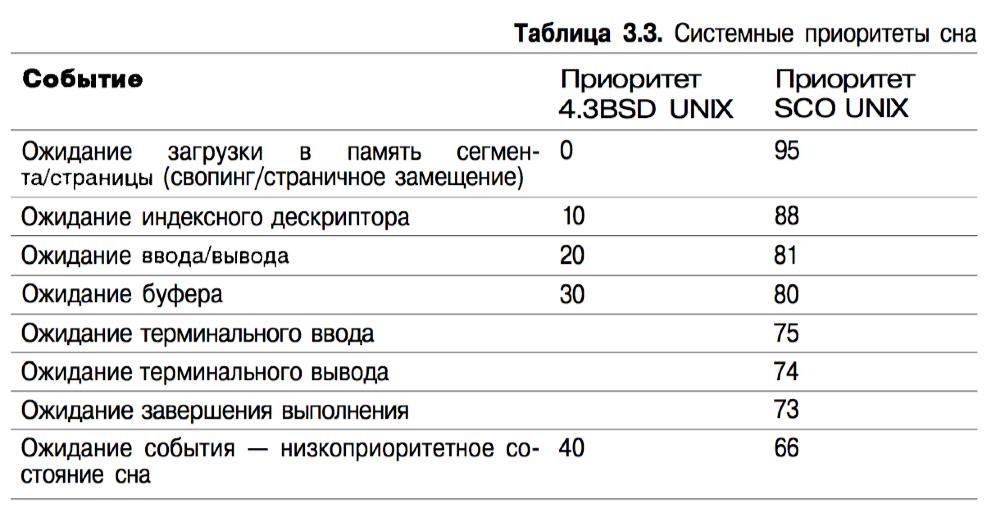
Когда процесс просыпается после блокирования в системном вызове, его приоритет будет временно повышен, чтобы дать ему предпочтение в режиме ядра. Поэтому планировщик использует *p\_usrpri* для хранения приоритета, назначаемого процессу при возврате в режим задачи, а *p\_pri* используется для хранения временного приоритета для выполнения в режиме ядра. Ядро системы связывает *приоритет сна* (величина, определяемая для ядра, поэтому лежит в пределах от 0 до 49) с событием или ожидаемым ресурсом, из-за которого процесс может быть заблокирован.

Когда процесс завершил выполнение системного вызова и находится в состоянии возврата в режим задачи, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи. Измененный таким образом приоритет может оказаться ниже, чем приоритет какого-либо иного запущенного процесса; в этом случае ядро системы произведет переключение контекста.

Приоритет в режиме задачи зависит от 2х факторов: «любезности» и последней измеренной величины использования процессора.

Степень любезности (nice value) – число от 0 до 39 (со значением 20 по умолчанию). Увеличение значения приводит к уменьшению приоритета. Фоновые процессы имею более высокие степени благоприятствия.

Процессу, ожидающему недоступного в данный момент ресурса, система определяет значение «*приоритет сна»,* выбираемое ядром из диапазона системных приоритетов и связанное с событием, вызвавшее это состояние.



Уменьшить эту величину для процесса может только «суперпользователь», так как при этом поднимется его приоритет. Степень любезности называется так из-за того, что какие-то пользователи могут быть поставлены в более выгодные условия другими пользователями, если те увеличат степень любезности для своих менее важных процессов.

Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы получили его примерно в равных количествах. Этот подход требует слежения за использованием процессора каждым из процессов. Поле *р\_срu* содержит величину результата последнего сделанного измерения использования процессора процессом. При создании процесса значение этого поля инициализируется нулем. На каждом тике обработчик таймера увеличивает *р\_срu* на единицу для текущего процесса до максимального значения, равного 127.

Каждую секунду ядро системы вызывает процедуру *schedcpu()* (запускается через отложенный вызов), которая уменьшает значение *р\_срu* каждого процесса исходя из фактора «полураспада» (*decay factor*).

В системе SVR3 фактор полураспада равен 0.5.

В 4.3 BSD для расчета фактора полураспада применяется следующая формула:

Процедура *schedcpu()* также пересчитывает приоритеты для режима задачи всех процессов по формуле:

где PUSER — базовый приоритет в режиме задачи, равный 50.

В результате, если процесс в последний раз использовал большое количество процессорного времени, его р\_срu будет увеличен. Это приведет к росту значения р\_usrpri и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его р\_срu, что приводит к повышению его приоритета.

**Пересчет динамических приоритетов для ОС семейств Windows**

Всего в ОС Windows 32 уровня запроса прерывания (от 0 до 31). Прерывания обслуживаются в порядке их приоритета. 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.

Никогда не повышаются приоритеты потоков реального времени (16–31). Повышение приоритета потока в Windows применяется только для потоков с приоритетом динамического диапазона (0-15). Но каким бы ни было приращение, приоритет потока никогда не будет больше 15. Таким образом, если к потоку с приоритетом 14 применить динамическое повышение на 5 уровней, то его приоритет станет равным только 15 (если приоритет потока равен 15, то повысить его нельзя).

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом, с той оговоркой, что конкретные, имеющие высокий приоритет и готовые к запуску потоки могут быть ограничены процессами, на которых им разрешено или предпочтительнее всего работать.

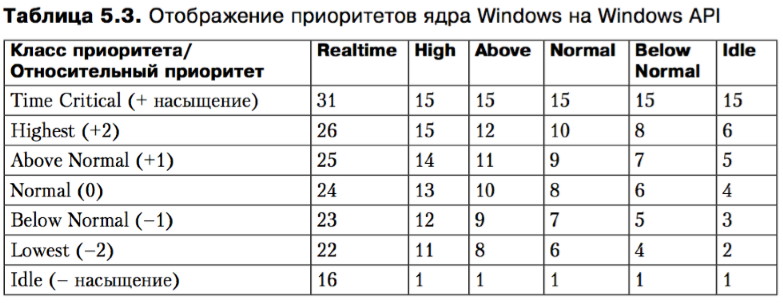
Также в Windows нет единого модуля или процедуры под названием «планировщик», код разбросан по ядру, где происходят события, связанные с планированием. Процедуры, выполняющие эти обязанности, обобщенно называются диспетчером ядра. Диспетчеризации потоков могут потребовать следующие события:

• Поток становится готовым к выполнению, например поток был только что создан или только что был освобожден от состояния ожидания;

• Поток выходит из состояния выполнения из-за окончания его кванта времени, его работа завершается, ему предоставляется возможность выполнения или он входит в состояние ожидания;

• Изменяется приоритет потока либо по причине вызова системной службы, либо по причине того, что Windows сама изменяет значение приоритета. Изменяется родственность процессора потока, и он больше уже не может быть запущен на том процессе, на котором выполнялся.

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, Высокий, Выше обычного, Обычный, Ниже обычного и Простоя. Затем назначается относительный приоритет отдельных потоков внутри этих процессов. Здесь номера представляют изменение приоритета, применяющееся к базовому приоритету процесса: Критичный по времени, Наивысший, Выше обычного, Обычный, Ниже обычного, Самый низший и Простоя.



**Случаи, когда текущие приоритеты потока могут повышаться**

1. После завершения операций ввода/вывода;

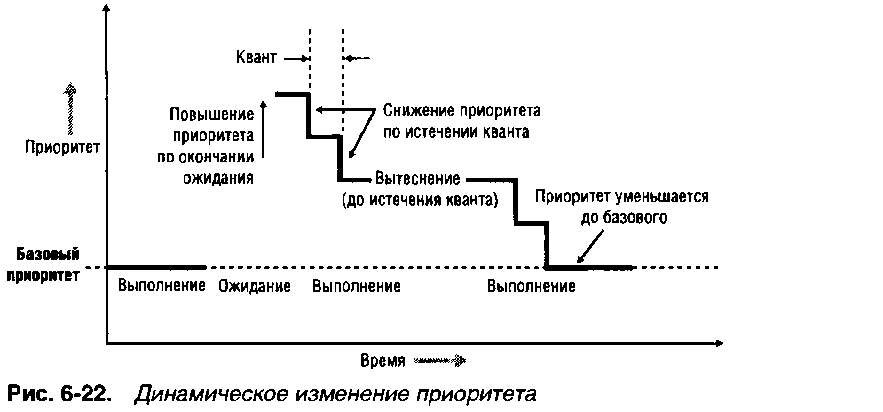
После завершения определенных операций ввода-вывода у потоков, ожидавших завершения таких операций, больше шансов немедленно возобновить выполнение (и обработать полученные данные). Необходимость повышения динамического приоритета после выполнения запроса на ввод/вывод указывает драйвер устройства через функцию ядра IoCompleteRequest. Важно, что для запросов на ввод/вывод, адресованных устройством с меньшим гарантированным временем отклика, предусматриваются большие приращения приоритета.

**Таблица рекомендованных приращений приоритета**

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

Приоритет потока повышается относительно базового уровня (а не текущего). На рис. 6-22 показано, что после динамического повышения приоритета поток выполняется с повышенным уровнем приоритета (в течении одного кванта), потом приоритет снижается на один уровень и потоку выделяется еще один квант. Так продолжается до тех пор, пока приоритет не станет равным базовому.

Но поток с более высоким приоритетом может вытеснить поток с повышенным приоритетом, но до понижения приоритета прерванный поток должен полностью отработать свой квант с повышенным приоритетом.



1. По окончании ожидания на событии или семафоре исполнительной системы;

Когда ожидание потока на событии исполнительной системы или объекте «семафор» успешно завершается (из-за вызова SetEvent, PulseEvent или Relea-seSemaphore), его приоритет повышается на 1 уровень.

Это целесообразно, так как потокам, блокируемым на событиях, процессорное время требуется реже, чем остальным (это позволяет равномернее распределять процессорное время).

1. По окончании операции ожидания потоками активного процесса

Всякий раз, когда поток в активном процессе завершает ожидание на объекте ядра, его текущий приоритет потока на величину текущего значения PsPrioritySeparation (это делает функция ядра KiUnwaitThread).

Это увеличивает отзывчивость интерактивного приложения по окончании ожидания и повышаются шансы на немедленное возобновление его потока.

1. При пробуждении GUI-потоков из-за операций с окнами

Приоритет потоков, владеющих окнами, дополнительно повышается на 2 уровня после их пробуждения из-за активности подсистемы управления окнами (например, при получении оконных сообщений). Подсистема управления окнами повышает приоритет, вызывая KeSetEvent для установки события, пробуждающего GUI-поток.

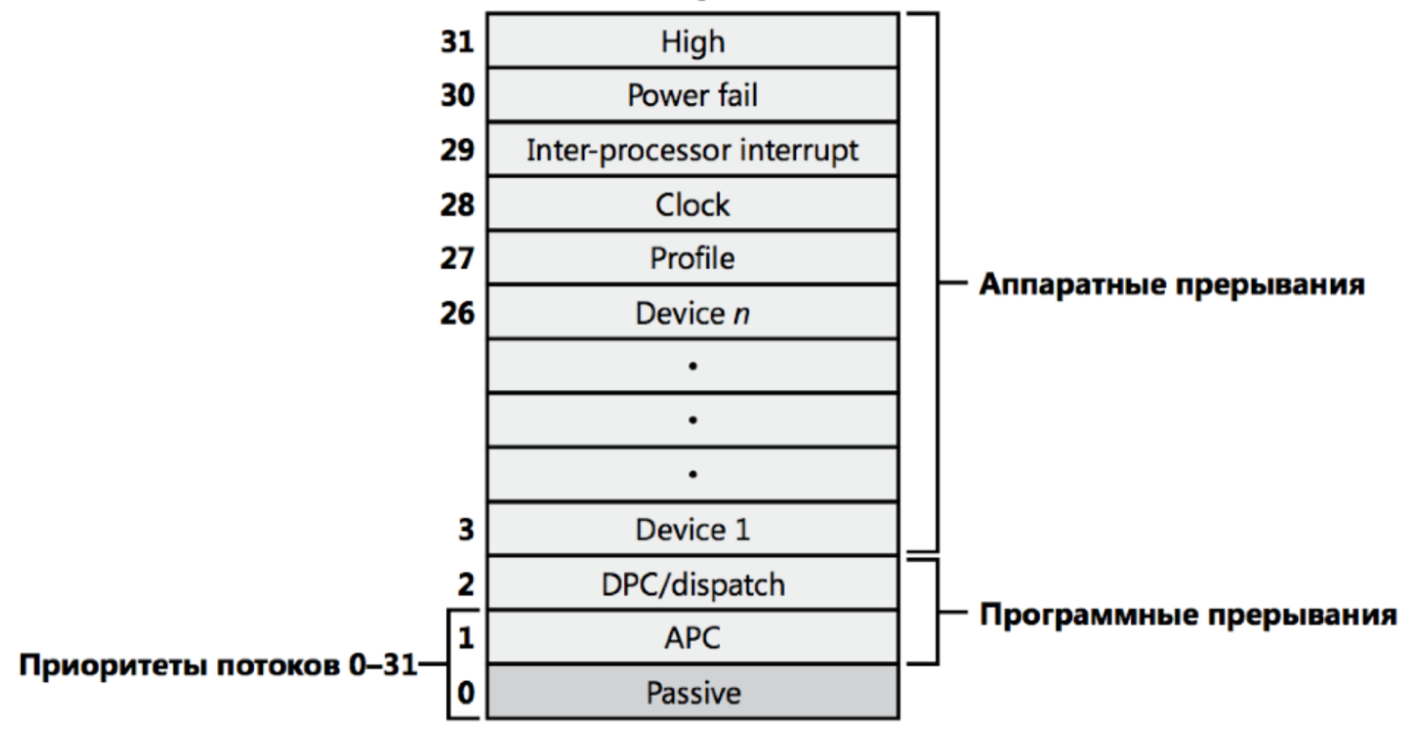
Это увеличивает отзывчивость интерактивного приложения по окончании ожидания и повышаются шансы на немедленное возобновление его потока.

1. Если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени

Раз в секунду диспетчер настройки баланса (системный поток, предназначенный главным образом для выполнения функций управления памятью), проверяет очереди готовых потоков и ищет потоки, которые находятся в состоянии готовности (Ready) в течение 4 секунд. Обнаружив такой поток, диспетчер настройки баланса повышает его приоритет до 15. В Windows 2000 и Windows ХР квант потока удваивается относительно кванта процесса. В Windows Server 2003 квант устанавливается равным 4 единицам. Как только квант истекает, приоритет потока немедленно снижается до исходного уровня. Если этот поток не успел закончить свою работу и если другой поток с более высоким приоритетом готов к выполнению, то после снижения приоритета он возвращается в очередь готовых потоков. В итоге через 4 секунды его приоритет может быть снова повышен.

Чтобы свести к минимуму расход процессорного времени, диспетчер настройки баланса сканирует лишь 16 готовых потоков. Если таких потоков с данным уровнем приоритета более 16, он запоминает тот поток, перед которым он остановился, и в следующий раз продолжает сканирование именно с него. Кроме того, он повышает приоритет не более чем у 10 потоков за один проход. Обнаружив 10 потоков, приоритет которых следует повысить (что говорит о необычайно высокой загруженности системы), он прекращает сканирование. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз.

**IRQL**

Для обеспечения поддержки мультизадачности системы, когда исполняется код режима ядра, Windows использует приоритеты прерываний IRQL. Прерывания обслуживаются в порядке их приоритета. При возникновении прерывания с высоким приоритетом процессор сохраняет информацию о состоянии прерванного потока и активизирует сопоставленный с данным прерыванием диспетчер ловушки. 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. 

Потоки, запущенные в режиме ядра, несмотря на изначальное планирование на пассивном уровне или уровне APC, могут поднять IRQL на более высокие уровни.

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. Поток выполняется на dispatch-уровне и выше, блокирует активность планировщика потоков и мешает контекстному переключению на своем процессоре.

Поток, запущенный в режиме ядра, может быть запущен на APC-уровне, если он запускает специальный APC-вызов ядра, или он может временно поднять IRQL до APC-уровня, чтобы заблокировать доставку специальных APC-вызовов ядра. Поток, выполняемый в режиме ядра на APC-уровне, может быть прерван в пользу потока с более высоким приоритетом, запущенным в пользовательском режиме на уровне passive.

**Вывод**

И в ОС Windows, и в UNIX обработчик системного таймера выполняет схожие основные функции:

1. обновление системного времени
2. уменьшение кванта процессорного времени, выделенного процессу
3. запуск планировщика задач
4. отправление отложенных вызовов на выполнение

Это обусловлено тем, что обе операционные системы являются системами разделения времени с вытеснением и динамическими приоритетами.