**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 (ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О т ч е т

# по лабораторной работе № 1 (часть 2)

**Название: Обработчик прерывания таймера**

# Дисциплина: Операционные системы

Студент гр. ИУ7-55Б О.Н.Талышева

(Подпись, дата) (И.О. Фамилия)

Преподаватель Н.Ю. Рязанова

(Подпись, дата) (И.О. Фамилия)

1. **Функции обработчика прерывания системного таймера в системах разделения времени**
   1. **Функции обработчика прерывания системного таймера в UNIX:**

* По тику:

1. Инкремент счетчика реального времени.
2. Инкремент счетчика времени с момента запуска системы (например, переменная lbolt в SVR4).
3. Инкремент счетчика процессорного времени, проведенного процессором в режиме задачи и режиме ядра.
4. Декремент счетчиков времени, с последующей установкой флагов для вызова обработчиков отложенных действий при достижении нулевого значения.

* По главному тику:

1. Инициализация отложенного вызова функции планировщика.
2. Инициализация отложенного вызова процедуры wakeup, переводящей процессы из состояния "спящий" в состояние "готовый к выполнению". В SVR4 это реализуется с помощью функции timeout(void (\*fn)(), caddr\_t arg, long delta), где:

* fn() — вызываемая функция.
* arg — аргументы для функции.
* delta — временной интервал в тиках до вызова функции.

1. Пробуждение системных процессов (таких как swapper и pagedaemon).
2. Уменьшение счетчика времени до отправки одного из следующих сигналов процессу:

* SIGALRM — по истечении заданного интервала реального времени.
* SIGPROF — по истечении времени профилирования.
* SIGVTALRM — по истечении времени, заданного в "виртуальном" таймере.
* По кванту:

1. Посылка текущему процессу сигнала SIGXCPU, если он превысил выделенное ему время использования процессора.
   1. **Функции обработчика прерывания системного таймера в Windows:**

* По тику:

1. Инкремент счётчика реального времени.
2. Декремент счётчиков отложенных задач (Deferred Procedure Calls).
3. Декремент кванта.

* По главному тику:

1. Инициализация диспетчера настройки баланса путём сброса объекта «событие», на котором он ожидает.

* По кванту:

1. Инициализация диспетчеризации потоков путём добавления соответствующего объекта DPC – Deferred procedure call (отложенный вызов процедуры) в очередь для выполнения задач перепланирования потоков.
2. **Пересчёт динамических приоритетов**

Операционные системы семейства Unix и Windows являются системами разделения времени с динамическими приоритетами и вытеснением. Динамические приоритеты могут иметь только пользовательские процессы, другие процессы имеют статические приоритеты.

* 1. **Unix**

В UNIX планирование процессов осуществляется на основе их приоритетов. Процесс с наивысшим приоритетом всегда получает доступ к процессору. Приоритеты пользовательских приложений могут корректироваться системой динамически в зависимости от текущей нагрузки, времени ожидания и состояния процесса.

Изначально ядро UNIX было невытесняющим, но современные версии перешли к вытесняющему ядру, что позволяет процессам с более высоким приоритетом вытеснять другие даже в режиме ядра. Это нововведение особенно важно для работы с задачами реального времени, такими как обработка мультимедиа.

Диапазон значений приоритетов

Приоритет процесса выражается целым числом от 0 до 127. Меньшие значения соответствуют более высоким приоритетам:

• 0–49 — зарезервированы для задач ядра, и их приоритеты неизменны.

• 50–127 — используются прикладными процессами, приоритет которых может изменяться.

Динамика приоритетов прикладных процессов определяется следующими факторами:

• Фактор "любезности" (p\_nice): принимает значения от 0 до 39 (по умолчанию 20). Чем ниже значение, тем выше приоритет.

• Показатель использования процессора (p\_cpu): отображает, насколько интенсивно процесс потреблял ресурсы процессора за последнее время.

Поля в дескрипторе процесса

Дескриптор процесса (proc) содержит информацию, связанную с приоритетами:

• p\_pri — текущий планировочный приоритет;

• p\_usrpri — приоритет процесса в пользовательском режиме;

• p\_cpu — степень использования процессора;

• p\_nice — параметр "любезности", задаваемый пользователем.

Динамическое перераспределение приоритетов

Система UNIX использует динамическое управление приоритетами, что помогает предотвратить бесконечное ожидание выполнения низкоприоритетных процессов.

Ежесекундно вызывается процедура schedcpu(), которая:

1. Обновляет p\_cpu, уменьшая его с учётом средней системной нагрузки (load average).

2. Пересчитывает p\_usrpri для прикладных процессов по формуле:

p\_usrpri = P\_USER + p\_cpu / 2 + 2 \* p\_nice,

где P\_USER = 50 — базовый пользовательский приоритет.

Приоритет сна

Когда процесс блокируется, например, ожидая ввода-вывода, ему присваивается "приоритет сна", связанный с событием блокировки. При пробуждении процессу присваивается p\_pri​, равное приоритету сна. Примеры значений приоритетов сна в системе 4.3BSD представлены в таблице 1.

Таблица 1 – Системные приоритеты сна

|  |  |  |
| --- | --- | --- |
| 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 |

Таким образом, современные версии UNIX реализуют вытесняющее планирование, что делает систему более отзывчивой для выполняемых процессов реального времени. Кроме того, динамическое перераспределение приоритетов даёт преимущество процессам с высокой активностью ввода-вывода и помогает избегать "застревания" низкоприоритетных задач.

**Потоки в системах семейства Unix**

Технология использования процессов обладает двумя важными ограничениями. Прежде всего, некоторым приложениям необходимо выполнять несколько крупных независимых друг от друга задач, при этом используя одно и то же адресное пространство, а также другие ресурсы. Традиционные системы UNIX либо выполняют отдельные задачи последовательно, либо применяют неуклюжие и малоэффективные механизмы поддержки параллельных таких операций.

Вторым ограничением применения традиционной модели является невозможность в полной мере задействовать преимущества многопроцессорных систем, так как процесс способен одновременно использовать только один процессор. Приложению приходится создавать несколько отдельных процессов и искать способы совместного использования памяти и ресурсов, а также синхронизации друг с другом.

Современные системы UNIX предлагают решение приведенных проблем при помощи различных технологий нитей, реализованных в ОС с целью поддерж­ки параллельного выполнения заданий. В каждом варианте UNIX для их обозначения применяются свои термины, например нити ядра, прикладные нити, прикладные нити, поддерживаемые ядром, C-threads, pthreads и «легковесные» процессы (lightweight processes).

* 1. **Windows**

В операционной системе Windows реализована вытесняющая система планирования с приоритетами, которая гарантирует, что всегда будет выполняться хотя бы один поток с наивысшим приоритетом. Каждый поток запускается на ограниченное время, называемое квант времени. Однако, если в это время появляется поток с более высоким приоритетом, текущий поток может быть вытеснен.

В Windows нет единого компонента под названием «планировщик». Вместо этого функции диспетчеризации выполняются различными процедурами, объединяемыми в понятие диспетчера ядра. Диспетчеризация может происходить в ответ на несколько событий:

* Поток становится готовым к выполнению.
* Поток завершает выполнение из-за истечения кванта времени.
* Поток завершает свою работу или переходит в состояние ожидания.
* Происходит изменение приоритета потока.
* Изменяется процессорная привязанность потока.

Система приоритетов в Windows включает 32 уровня, варьирующихся от 0 до 31. Эти уровни делятся следующим образом:

* Шестнадцать уровней реального времени (16–31).
* Шестнадцать уровней с изменяемым приоритетом (0–15), при этом уровень 0 зарезервирован для потоков, отвечающих за обнуление страниц.

Приоритеты потоков устанавливаются из двух источников: Windows API и ядра Windows. При создании процесса ему назначается класс приоритета из следующих категорий:

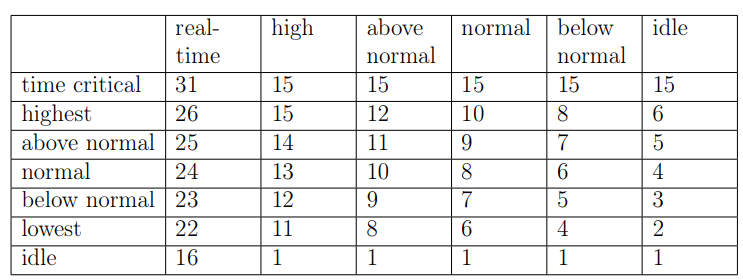
* Реальное время (real-time, 4).
* Высокий (high, 3).
* Выше обычного (above normal, 7).
* Обычный (normal, 2).
* Ниже обычного (below normal, 5).
* Простой (idle, 1).

После этого определяется относительный приоритет потоков внутри процессов, который может варьироваться следующим образом:

* 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).
* Простоя (idle, -15).

Критичные уровни по времени и простоя (15 и -15) считаются уровнями насыщения и представляют собой фиксированные приоритеты, тогда как относительный приоритет — это смещение от базового приоритета процесса.

Таблица 2 – Соответствие между приоритетами Windows API и ядра Windows.



Процесс имеет только одно базовое значение приоритета, тогда как каждый поток обладает двумя значениями: динамическим (текущим) и базовым. При планировании используется именно текущее значение приоритета. В определённых случаях Windows временно повышает приоритет потоков, чтобы улучшить отзывчивость системы и уменьшить задержки. Данное повышение происходит мгновенно и может повлиять на распределение задач процессора.

Если поток использует весь свой квант времени, его приоритет понижается на один уровень. При повторном исчерпании кванта, он снова снижается, пока не достигнет своего базового уровня. Таким образом, система поддерживает баланс между производительностью и отзывчивостью потоков.

Cценарии повышения приоритета:

* Повышение вследствие событий планировщика или диспетчера (сокращение задержек).

Повышение вследствие завершения ввода-вывода (сокращение задержек). Рекомендуемые значения повышения приоритета приведены в таблице 3.

* Повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика).
* Повышение вследствие слишком продолжительного ожидания ресурса ис полняющей системы (предотвращение зависания).
* Повышение в случае, когда готовый к запуску поток не был запущен в течение определенного времени (предотвращение зависания и смены приоритетов)
* Повышение приоритета владельца блокировки.
* Повышение вследствие ожидания объекта ядра.
* Повышение приоритета потоков первого плана после ожидания (улучшение отзывчивости интерактивных приложений).
* Повышение приоритета после пробуждения GUI-потока (потоки-владельцы окон получают при пробуждении дополнительное повышение приоритета на 2).
* Повышения приоритета, связанные с перезагруженностью центрального процессора (CPU Starvation).
* Другие псевдоповышающие механизмы, проявляющие себя при проигрывании мультимедиа. В отличие от других повышений приоритета, эти механизмы применяются непосредственно в режиме ядра. Повышение приоритета проигрывания мультимедиа управляются службой 8 планировщика класса мультимедиа MMCSS (это не является настоящим повышением, служба просто устанавливает по необходимости новые базовые приоритеты для потоков).

Таблица 3 – Рекомендуемые значения повышения приоритета

|  |  |
| --- | --- |
| Устройство | Повышение приоритета |
| Жесткий диск, компакт-диск, видеоустройство | 1 |
| Сеть, почтовый слот, именованный канал | 2 |
| Клавиатура, мышь | 6 |
| Звуковое устройство | 8 |

**MMCSS**

Для мультимедийных приложений, таких как Windows Media Player или 3D-игры, стандартные повышения приоритетов операционной системы Windows недостаточны для обеспечения требований к ресурсам. Это может привести к проблемам, таким как перескакивания или звуковые дефекты, что ухудшает пользовательский опыт.

Чтобы справиться с этими проблемами, в клиентских версиях Windows была внедрена служба MMCSS (Multimedia Class Scheduler Service). Эта служба гарантирует стабильное воспроизведение мультимедийного контента без сбоев. MMCSS работает с конкретными задачами, такими как:

* Аудио,
* Захват,
* Распределение,
* Игры,
* Профессиональное аудио,
* Задачи администратора многозадачного режима.

Настройки MMCSS, включая список задач, могут быть изменены поставщиками программного обеспечения. Параметр SystemResponsiveness в реестре позволяет регулировать время, выделенное для потоков с низким приоритетом.

Каждая задача имеет свой категории планирования, что определяет приоритет потоков, зарегистрированных с MMCSS. Разные категории планирования включают:

Таблица 4 – Категории планирования

|  |  |  |
| --- | --- | --- |
| Категория | Приоритет | Описание |
| High (Высокая) | 23–26 | Потоки профессионального аудио, с приоритетом выше, чем у других потоков (кроме системных) |
| Medium (Средняя) | 16–22 | Потоки приложений первого плана, например Windows Media Player |
| Low (Низкая) | 8–15 | Все остальные потоки, не относящиеся к предыдущим категориям |
| Exhausted (Исчерпавшие потоков) | 1–7 | Потоки, исчерпавшие свою долю времени процессора, выполняются только при отсутствии более приоритетных потоков |

Механизм MMCSS повышает приоритет потоков до уровня соответствующей категории, а затем понижает их до категории Exhausted, чтобы другие потоки также получали время для выполнения.

По умолчанию, мультимедийные потоки получают 80% времени процессора, остальные потоки — 20%. Сама служба MMCSS выполняется с приоритетом 27, чтобы вытеснить потоки Pro Audio, понижая их приоритет до Exhausted.

Несмотря на повышение приоритета, планировщик операционной системы продолжает управлять этими потоками, что позволяет почти непрерывное выполнение мультимедийных приложений.

**IRQL**

IRQL (уровень запроса прерывания) — это механизм, используемый в Windows для управления приоритетами прерываний и обеспечения мультизадачности в системе при выполнении кода в режиме ядра. Прерывания обрабатываются в порядке их приоритета: когда происходит прерывание с более высоким приоритетом, процессор сохраняет состояние текущего потока и активирует соответствующий обработчик прерывания (ISR — Interrupt Service Routine).

Обработчик прерывания повышает уровень IRQL, выполняет необходимые действия и затем понижает IRQL до предыдущего значения, восстанавливая сохраненное состояние машины. После этого прерванный поток продолжает выполнение с того места, где был остановлен.

Когда IRQL понижается, могут начать обрабатываться ранее замаскированные прерывания с более низким приоритетом, что запускает аналогичный процесс обработки.

Уровни IRQL в ядре представлены номерами от 0 до 31, при этом более высокие номера соответствуют более высоким приоритетам прерываний. На рисунке 2 изображены различные уровни IRQL.

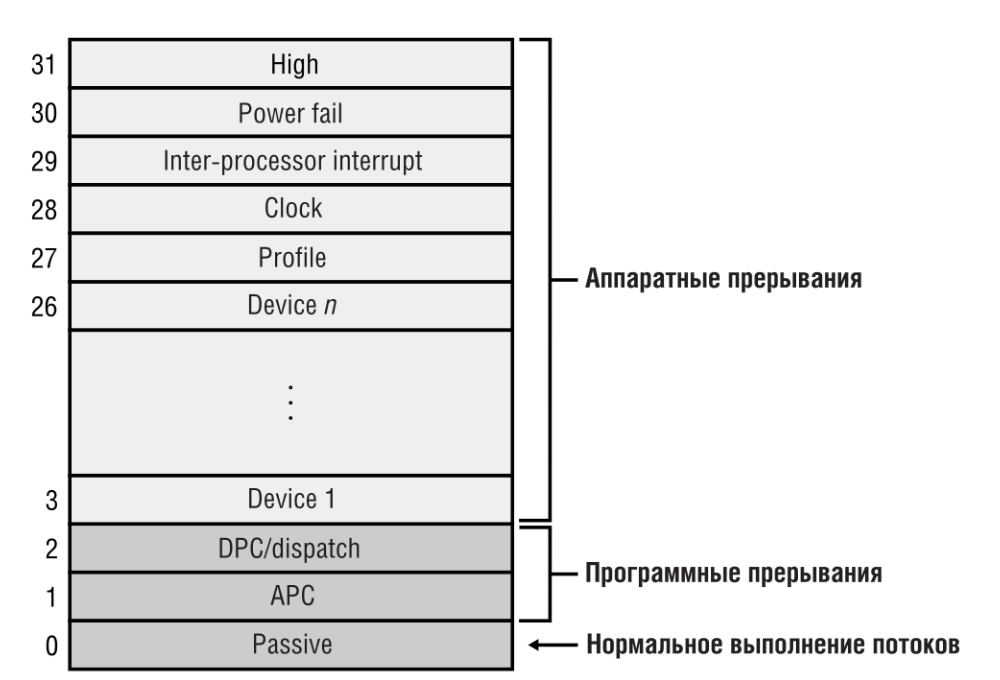


Рисунок 1 – Уровни запросов прерываний

1. **Выводы**

Операционные системы семейств Unix и Windows являются системами разделения времени с вытеснением и динамическими приоритетами, поэтому обработчик прерывания от системного таймера в них выполняет схожие функции:

* инкремент счётчика системного времени;
* декремент кванта;
* инициализация отложенных действий.

Пересчет динамических приоритетов реализуется только для пользовательских процессов с целью предотвращения бесконечного откладывания их выполнения.