|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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** “Функции обработчика прерывания системного таймера и пересчет динамических приоритетов”  **Студент** Блохин Д.М.  **Группа** ИУ7-52Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н. Ю. |  |

Москва.

2020 г.

**Функции обработчика прерывания от системного таймера в защищенном режиме для ОС Unix/Linux.**

По тику:

* инкремент часов и других таймеров системы;
* декремент кванта;
* декремент счётчика времени, оставшегося до выполнения отложенного вызова.

По главному тику:

* декремент счётчиков времени, оставшихся до посылки ядром процессу одного из сигналов (будильники):
  + SIGLARM – если будильник был установлен с помощью *будильника реального времени*. Посылается через указанный промежуток реального времени;
  + SIGPROF - если будильник был установлен с помощью *будильника профиля процесса*, который измеряет время работы процесса;
  + SIGVTALAR если будильник был установлен с помощью *будильника виртуального времени,* которыйследиттолько за количеством времени работы процесса в режиме задачи.
* пробуждение системных процессов (например, pagedaemon, swapper) (под "пробуждением" имеется в виду инициализация отложенного вызова процедуры "wakeup", которая перемещает дескрипторы процессов из списка спящих в очередь готовых к выполнению);

По кванту:

* посылка текущему процессу сигнала SIGXCPU, если тот превысил выделенную ему квоту использования процессора.

**Функции обработчика прерывания от системного таймера в защищенном режиме для ОС Windows.**

По тику:

* инкремент счётчика системного времени;
* декремент счётчика отложенных задач;
* декремент кванта текущего потока.

По главному тику:

* Освобождение объекта “событие”, которое ожидает диспетчер баланса, по сигналу таймера, срабатывающего раз в секунду. Диспетчер настройки баланса же по событию от таймера сканирует очередь готовых процессов и повышает приоритет процессов.

По кванту:

* инициация диспетчеризации потоков (добавление соответствующего объекта в очередь DPC).

**Пересчет динамических приоритетов в Unix/Linux:**

Традиционное ядро Unix является строго невытесняющим. Если процесс выполняется в режиме ядра, то ядро не заставит такой процесс уступить процессор более высокоприоритетному процессу. Однако современные ядра являются вытесняемыми для того, чтобы было система имела возможность обслуживать процессы реального времени, например, проигрывание музыки.

Приоритет процесса 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, следовательно, прикладные процессы могут обладать приоритетом в диапазоне 50-127.

Структура proc содержит следующие поля, относящиеся к приоритетам:

* p\_pri – текущий приоритет планирования;
* p\_usrpri – приоритет режима задачи;
* p\_cpu – результат последнего измерения использования процессора;
* p\_nice – фактор «любезности», устанавливаемый пользователем.

Планировщик использует p\_рri для принятия решения о том, какой процесс направить на выполнение. Когда процесс находится в режиме задачи, значение его p\_рri идентично p\_usrpri. Когда процесс просыпается после блокирования в системном вызове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра. Следовательно, планировщик использует p\_usrpri для хранения приоритета, который будет назначен процессу при возврате в режим задачи, a p\_рri — для хранения временного приоритета для выполнения в режиме ядра.

Когда замороженный процесс просыпается, ядро устанавливает значение его p\_pri, равное приоритету сна события или ресурса. Поскольку приоритеты ядра выше, чем приоритеты режима задачи, такие процессы будут назначены на выполнение раньше, чем другие, функционирующие в режиме задачи. Такой подход позволяет системным вызовам быстро завершить свою работу, что является желательным, так как процессы во время выполнения вызова могут занимать некоторые ключевые ресурсы системы, не позволяя пользоваться ими другим.

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 процесса в режиме задачи зависит от «любезности» (nice) и последней измеренной величины использования процессора. Степень любезности (nice value) является числом в диапазоне от 0 до 39 со значением 20 по умолчанию. При увеличении значения любезности приоритет процесса уменьшается. Фоновым процессам автоматически задаются более высокие значения «любезности». Уменьшить эту величину для какого-либо процесса может только суперпользователь, поскольку при этом поднимется его приоритет.

Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы получили его примерно в равных количествах. Такой подход требует слежения за использованием процессора каждым из процессов. Поле р\_срu структуры рrос 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. Более того, система с помощью отложенного вызова каждую секунду вызывает процедуру schedcpu(), которая уменьшает значение р\_срu каждого процесса исходя из фактора «полураспада» (decay factor) (в системе SVR3 фактор полураспада равен ½). В 4.3BSD для расчета фактора полураспада применяется следующая формула:

Где load\_average – это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

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 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 р\_сри, что приводит к повышению его приоритета. Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов.

**Пересчет динамических приоритетов в Windows:**

При запуске процесса ему назначается приоритет, который в Windows называется базовым. Приоритет потока вычисляется относительно базового приоритета процесса.

В Windows реализуется вытесняющая, приоритетная система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом. Windows осуществляет планировку потоков по вытесняющему принципу. Т.е. поток с более низким приоритетом может быть вытеснен ещё до окончания выделенного ему кванта времени.

Код, отвечающий за планирование, реализован в самом ядре. В Windows отсутствует модуль или процедура под названием «планировщик». Набор процедур, выполняющих указанные функции, называется диспетчером ядра.

Само же планирование вызывается при следующих условиях:

* выполняющийся поток блокируется на мьютексе, семафоре, событии, вводе/выводе и т.д;
* поток подаёт сигнал об объекте;
* истекает квант времени потока.

В Windows используется 32 уровня приоритета в диапазоне от 0 до 31, из которых:

* от 0 до 15 – динамические уровни (уровень 0 зарезервирован для потока обнуления страниц);
* от 16 до 31 – уровни реального времени.

Уровни приоритета потоков назначаются с двух позиций: от Windows API и от ядра Windows.

Сначала Windows API систематизирует процессы по базовому приоритету:

* Реального времени – Real-time (4);
* Высокий – High (3);
* Выше обычного – Above Normal (6);
* Обычный – Normal (2);
* Ниже обычного – Below Normal (5);
* Уровень простоя – Idle (1);

Далее представлены приоритеты отдельных потоков внутри процессов. Здесь числа представляют приращение, применяемое к базовому приоритету процессора:

* Критический 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);
* Уровень простоя – Idle (-15);

Уровень, критичный по времени, и уровень простоя (+15 и -15) называются уровнями насыщения и представляют конкретные применяемые уровни вместо смещений.

Планировщик Windows периодически настраивает текущий приоритет потоков используя внутренний механизм повышения приоритета. Во многих случаях это делается для уменьшения различных задержек и ускорения отклика. В других случаях это повышение применяется для предотвращения зависаний и инверсии приоритетов.

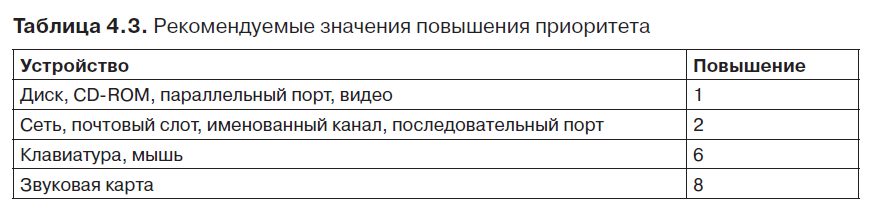
Соответствие между Windows-приоритетами и внутренними номерными приоритетами Windows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Класс приоритета/  относительный приоритет | Realtime | High | Above-Normal | 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  (- Saturation) | 16 | 1 | 1 | 1 | 1 | 1 |

Планировщик Windows периодически настраивает текущий приоритет потоков, используя внутренний механизм повышения приоритета. Повышение приоритета вступает в действие немедленно и может вызвать изменения в планировании процессора. Во многих случаях это делается для уменьшения различных задержек (чтобы потоки быстрее реагировали на события, в ожидании которых они находятся) и ускорения отклика. В других случаях это повышение применяется для предотвращения зависаний и инверсии приоритетов.

Ряд сценариев повышения приоритета потока:

* повышение вследствие событий планировщика или диспетчера (сокращение задержек);
* повышение вследствие завершения ввода/вывода (сокращение задержек);
* повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика);
* повышение вследствие слишком продолжительного ожидания ресурса исполняющей системы (ERESOURCE) (предотвращение зависания);
* повышение в случае, когда готовый к запуску поток не был запущен в течение определенного времени (предотвращение зависания и инверсии приоритетов).

Рекомендуемые приращения приоритета: 

Стоит отметить, что Windows никогда не повышает приоритет потоков в диапазоне реального времени (от 16 до 31). Поэтому планирование относительно других потоков в диапазоне реального времени всегда предсказуемо. Windows предполагает, что при использовании приоритетов потоков реального времени вы знаете, что делаете.

Стоит отметить, что текущий приоритет потока в динамическом диапазоне может быть понижен до базового приоритета путем вычитания всех повышений.

Как уже было отмечено, Windows временно повышает приоритет потоков по окончании определенных операций ввода-вывода, поэтому у потоков, которые ожидают ввод/вывод, по окончании ожидания гораздо больше шансов сразу же возобновить своё выполнение и обработать полученные данные. Драйвер устройства указывает через функцию ядра IoCompleteRequest на необходимость динамического повышения приоритета после выполнения запроса на ввод-вывод. Для запросов на ввод-вывод, адресованных устройствам, которые гарантируют меньшее время отклика, предусматриваются большие приращения приоритета.

|  |  |
| --- | --- |
| Устройство | Приращение |
| Звуковая карта | 8 |
| Клавиатура, мышь | 6 |
| Сеть/именованный канал/последовательный порт/почтовый слот | 2 |
| CD-ROM/параллельный порт/диск/видео | 1 |

Перескакивания и другие звуковые дефекты были в прошлом обычным источником раздражения со стороны пользователей Windows, а имеющийся в Windows аудиостек пользовательского режима только усугублял ситуацию, поскольку он еще больше повышал шансы на вытеснение потоков. Для решения этой проблемы клиентские версии Windows используют драйвер MMCSS

Клиентские приложения регистрируются у MMCSS, вызывая функцию AvSetMmThreadCharacteristics с именем задачи, которое должно соответствовать одному из подразделов HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ Multimedia\SystemProfile\Tasks. (OEM-производители могут изменять этот список, включая в него необходимые задачи.) В исходном варианте существуют следующие задачи:

* аудио;
* захват;
* распределение;
* игры;
* низкая задержка;
* воспроизведение;
* аудио профессионального качества;
* задачи администратора многооконного режима.

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) — это основной фактор, определяющий приоритет потоков, зарегистрированных с MMCSS

|  |  |  |
| --- | --- | --- |
| Категория | Приоритет | Описание |
| High | 23-26 | Потоки Pro Audio, выполняемые с приоритетом выше, чем у других потоков в системе, за исключением критических системных потоков |
| Medium | 16-22 | Потоки, являющиеся частью приложений первого плана, например Windows Media Player |
| Low | 8-15 | Все остальные потоки, не являющиеся частью предыдущих категорий |
| Exhausted | 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 |

Механизм, положенный в основу MMCSS, повышает приоритет потоков внутри зарегистрированного процесса до уровня, соответствующего их категории планирования и относительного приоритета внутри этой категории на гарантированный период. Затем он снижает категорию этих потоков до Exhausted, чтобы другие, не относящиеся к мультимедийным приложениям потоки также получили шанс на выполнение.

Стоит отметить, что Windows также включен общий механизм ослабления загруженности центрально­го процессора, который называется ***диспетчером настройки баланса****.*

Один раз в секунду диспетчер настройки баланса (системный поток, предназначенный главным образом для выполнения функций управления памятью), сканирует очередь готовых потоков и ищет потоки, которые находятся в состоянии ожидания в течение 4 секунд. При обнаружении такого потока, диспетчер настройки баланса повышает его приоритет до 15 единиц. 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. В случае, если этот поток не успел закончить свою работу и есть другой готовый к запуску поток с более высоким приоритетом, то поток с пониженным приоритетом возвращается в очередь готовых потоков, где он становится подходящим для ещё одного повышения, если будет оставаться в очереди следующие 4 секунды. Для минимизации затрачиваемого на его работу времени центрального процессора, 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. Важно уточнить, что повышения приоритета происходить 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.

**Вывод**

Функции обработчика прерывания в операционных системах Windows и Unix/Linux имеют схожие черты, так как обе эти операционные системы являются системами разделения времени с динамическими приоритетами и вытеснением. Общие функции обработчика:

* инициализация отложенных действий, относящихся к работе планировщика (например, пересчёт приоритетов);
* счет тиков системного времени;
* выполнение декремента кванта времени;
* выполнение декремента счётчиков времени.

Обе операционные системы выполняют пересчет приоритетов пользовательских процессов. Делается это для исключения бесконечного откладывания и повышения отзывчивости системы на действия пользователя.

Следует отметить, что как в Unix/Linux, так и в Windows пересчёт приоритетов возможен только для пользовательских процессов.