|  |  |
| --- | --- |
| **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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

**О Т Ч Ё Т**

**по лабораторной работе №1 (часть 2)**

**Тема:** «Функции обработчика прерывания системного таймера и пересчет динамических приоритетов»

**Студент** Петрова А. А.

**Группа** ИУ7-56Б

**Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Преподаватель** Рязанова Н.Ю.

**1. Функции обработчика прерывания от системного таймера в защищённом режиме**

**1.1. Unix**

**Функции обработчика прерывания от системного таймера, выполняющиеся по тику:**

* инкремент счётчика тиков аппаратного таймера;
* декремент кванта текущего потока;
* инкремент поля p\_cpu дескриптора текущего процесса до максимального значения, равного 127;
* инкремент часов и других таймеров системы;
* декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов.

**По главному тику:**

* инициализация отложенных вызовов функций, относящихся к работе планировщика, таких как пересчёт приоритетов;
* пробуждение в нужные моменты системных процессов swapper и pagedaemon. Пробуждение обозначает инициализацию отложенного вызова процедуры wakeup, которая перемещает дескрипторы процессов из списка «спящих» в очередь готовых к выполнению;
* декремент счётчика времени, оставшегося до отправления 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.

**По кванту:**

* посылка текущему процессу сигнала **SIGXCPU**, если тот превысил выделенный ему квант использования процессора. По получению сигнала обработчик сигнала прерывает выполнение процесса.

**1.2. Windows**

**Функции обработчика прерывания от системного таймера, выполняющиеся по тику:**

* инкремент счётчика системного времени;
* декремент кванта текущего потока;
* декремент счётчиков времени отложенных задач;
* в случае, если активен механизм профилирования ядра, происходит инициализация отложенного вызова обработчика ловушки профилирования ядра путём добавления объекта в очередь DPC: обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

**По главному тику:**

* освобождение объекта «событие», которое ожидает диспетчер настройки баланса. Диспетчер настройки баланса по событию от таймера сканирует очередь готовых процессов и повышает приоритет процессов, которые находились в состоянии ожидания дольше 4 секунд.

**По кванту:**

* инициализация диспетчеризации потоков – добавление соответствующего объекта в очередь DPC.

**2. Пересчёт динамических приоритетов**

**В ОС семейств UNIX и WINDOWS могут пересчитываться только приоритеты пользовательских процессов.**

**2.1. Unix**

В современных системах UNIX ядро является вытесняющим. Это значит, что процесс в режиме ядра может быть вытеснен боле 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:

* 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 изменён 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 загруженности 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 вернётся в режим задачи.

Когда процесс просыпается после блокирования в системном вызове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра.

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.

Процессу, ожидающему недоступный в данный момент ресурс, система определяет значение, выбираемое ядром из диапазона системных приоритетов и связанное с событием, вызвавшее это состояние.

В таблице ниже приведены значения приоритетов сна для систем UNIX и SCO UNIX 5.0. Направление роста значений приоритета для этих систем различно — в BSD UNIX большему значению соответствует более низкий приоритет.

|  |  |  |
| --- | --- | --- |
| **Событие** | **Приоритет 4.3BSD UNIX** | **Приоритет 4.3SCO 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 |

При создании процесса поле 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 schedcpy() , которая уменьшает значение p\_cpu каждого процесса, исходя из фактора «полураспада». В системе 4.3BSD фактор полураспада рассчитывается по формуле:

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

Приоритеты для режима задачи всех процессов в процедуре schedcpy() пересчитываются по формуле:

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 приведёт к понижению приоритета. Чем дольше процесс простаивает в очереди на исполнение, тем больше фактор полураспада 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 противовес процессам, производящим много вычислений.

**2.2. Windows**

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 потоков, готовых к выполнению.

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 завершён за квант времени или был вытеснен потоком с более высоким приоритетом, то после снижения приоритета поток возвращается в очередь потоков, готовых к выполнению.

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 загруженности системы.

Windows использует 32 уровня приоритета, которые описываются целыми числами от 0 до 31, где 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 Windows. 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 внутри этих 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.

При назначении потоку уровня «критичный по времени» поток получает наивысший возможный для класса его процесса приоритет:

* 31 — для потока процесса уровня реального времени
* 15 — для потока процесса динамического уровня

При назначении потоку уровня простоя поток получает самый низкий возможный для класса его процесса приоритет:

* 16 — для потока процесса уровня реального времени
* 1 — для потока процесса динамического уровня

Поток характеризуют текущее (динамическое) и базовое значения приоритета.

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

* **событий планировщика или диспетчера (сокращение задержек)**

при наступлении события диспетчера вызываются процедуры с целью проверить не должны ли на локальном процессоре быть намечены какиелибо потоки, которые не должны быть спланированы.

* **завершения ожидания**

повышения приоритета, связанные с завершением ожидания (unwait boosts), пытаются уменьшить время задержки между потоком, пробуждающимся по сигналу объекта (переходя тем самым в состояние Ready), и потоком, фактически приступившим к своему выполнению в процессе, который не находился в состоянии ожидания (переходя тем самым в состояние Running).

* **повышения приоритета владельца блокировки**

поскольку блокировки ресурсов исполняющей системы (ERESOURCE) и блокировки критических разделов используют основные объекты диспетчеризации, в результате освобождения этих блокировок осуществляются повышения приоритета, связанные с завершением ожидания. С другой стороны, поскольку высокоуровневые реализации этих объектов отслеживают владельца блокировки, ядро может принять более взвешенное решение о том, какого вида повышение должно быть применено с помощью AdjustBoost.

* **завершения ввода-вывода (сокращение задержек)**

Windows даёт временное повышение приоритета при завершении определенных операций ввода-вывода, при этом потоки, ожидавшие ввода-вывода, имеют больше шансов сразу же запуститься и обработать то, чего они ожидали.

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового путем вычитания всех его повышений.

В таблице ниже приведены рекомендуемые значения повышения приоритета для устройств ввода-вывода.

|  |  |
| --- | --- |
| **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 всегда повышается относительно базового, а не текущего уровня.

* **Повышение при ожидании ресурсов исполняющей системы**

Когда поток пытается получить ресурс исполняющей системы (ERESOURCE), который уже находится в исключительном владении другого потока, он должен войти в состояние ожидания до тех пор, пока другой поток не освободит ресурс. Для ограничения риска взаимных исключений исполняющая система выполняет это ожидание, не входя в бесконечное ожидание ресурса, а интервалами по пять секунд. Если по окончании этих пяти секунд ресурс все еще находится во владении, исполняющая система пытается предотвратить зависание центрального процессора путем завладения блокировкой диспетчера, повышения приоритета потока или потоков, владеющих ресурсом, до значения 14 (если исходный приоритет владельца был меньше, чем у ожидающего, и не был равен 14), перезапуска их квантов и выполнения еще одного ожидания.

* **Повышение приоритета потоков первого плана после ожидания**

Смысл этого повышения заключается в улучшении отзывчивости интерактивных приложений: если дать приложениям первого плана небольшое повышение приоритета при завершении ожидания, у них повышаются шансы сразу же приступить к работе, особенно когда другие процессы с таким же базовым приоритетом могут быть запущены в фоновом режиме.

* **Повышение приоритета после пробуждения GUI-потока**

Потоки-владельцы окон получают при пробуждении дополнительное повышение приоритета на 2 из-за активности системы работы с окнами, например поступление сообщений от окна. Система работы с окнами (Win32k.sys) применяет это повышение приоритета, когда вызывает функцию KeSetEvent для установки события, используемого для пробуждения GUI-потока. Смысл этого повышения похож на смысл предыдущего повышения — содействие интерактивным приложениям.

* **Повышения приоритета, связанные с перезагруженностью центрального процессора**

В Windows также включен общий механизм ослабления загруженности центрального процессора, который называется диспетчером настройки баланса и является частью потока (речь идет о системном потоке, который существует главным образом для выполнения функций управления памятью). Один раз в секунду этот поток сканирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ожидания (то есть не были запущены) около 4 секунд. Если такой поток будет найден, диспетчер настройки баланса повышает его приоритет до 15 единиц и устанавливает квантовую цель эквивалентной тактовой частоте процессора при подсчете 3 квантовых единиц. Как только квант истекает, приоритет потока тут же снижается до обычного базового приоритета. Если поток не был завершен и есть готовый к запуску поток с более высоким уровнем приоритета, поток с пониженным приоритетом возвращается в очередь готовых потоков, где он опять становится подходящим для еще одного повышения приоритета, если будет оставаться в очереди следующие 4 секунды.

* **Повышение приоритетов для мультимедийных приложений и игр**

В клиентские версии Windows была внедрена служба MMCSS, чьей целью было гарантировать проигрывание мультимедийного контента приложений, зарегистрированных с этой службой, без каких-либо сбоев. MMCSS работает с вполне определенными задачами, включая следующие:

* + аудио
  + аудио профессионального качества
  + захват изображения с экрана
  + игры
  + воспроизведение медиаконтента
  + задачи администратора многооконного режима

Функции MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, соответствующего их категориям планирования. Затем, их приоритет снижается до уровня, соответствующего категории Exhausted, чтобы другие потоки также могли получить ресурс.

Windows никогда не повышает приоритет потоков в диапазоне реального времени (16-31).

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового приоритета путем вычитания всех повышений.

**Вывод**

Для семейств ОС Windows и Unix/Linux функции обработчика прерывания от системного таймера в защищенном режиме схожи, так как эти ОС являются системами разделения времени.

Общие функции обработчика:

* Инициализируют отложенные действия, относящиеся к работе планировщика, такие как пересчет приоритетов
* Выполняют декремент счетчиков времени: часов, таймеров, будильников реального времени, счетчиков времени отложенных действий
* Выполняют декремент кванта.