|  |  |
| --- | --- |
| 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)** |

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 01 |
|  |  |

*Название:* ***Изучение прерывания от системного таймера (2 часть)***

# *Дисциплина*: «Операционные системы»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ***ИУ7И-56Б*** |  |  | **Нгуен Ф. С.** |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | **Рязанова Н. Ю.** |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

1. **Функции обработчика прерываний от системного таймера**
   1. **Windows**
      1. ***по тику***
         * Инкремент счетчика системного времени.
         * Декремент счетчиков отложенных задач.
         * Декремент остатка кванта текущего потока.
         * Активация обработчика ловушки профилирования ядра.
      2. ***по главному тику***
         * Инициализация диспетчера настройки баланса путем освобождения объекта «событие> на котором он ожидает.
      3. ***по кванту***
         * Инициализация диспетчеризации потоков добавлением соответствующего объекта DPC в очередь.
   2. **UNIX**
      1. ***по тику***
         * Инкремент счетчика использования процессора текущим процессом.
         * Инкремент часов и других таймеров системы.
         * Декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов и отправка отложенных вызовов на выполнение, при достижении нулевого значения счетчика.
         * Декремент кванта.
      2. ***по главному тику***
         * Добавление в очередь отложенных вызовов функций планировщика.
         * Пробуждение системных процессов 𝑠𝑤𝑎𝑝𝑝𝑒𝑟 и 𝑝𝑎𝑔𝑒𝑑𝑎𝑒𝑚𝑜𝑛.
         * Декремент счетчиков времени, оставшегося до отправления сигналов тревоги:
           + **𝑆𝐼𝐺𝐴𝐿𝑅𝑀** - сигнал будильника реального времени, который отправляется по истичении заданного промежутка реального времени;
           + **𝑆𝐼𝐺𝑃𝑅𝑂𝐹** - сигнал будильника профиля процесса, который измеряет время работы процесса;
           + **𝑆𝐼𝐺𝑉𝑇𝐴𝐿𝑅𝑀** - сигнал будильника виртуального времени, который измеряет время работы процесса в режиме задачи
      3. ***по кванту***
         * При превышении текущим процессом выделенного кванта, отправка сигнала **𝑆𝐼𝐺𝑋𝐶𝑃𝑈** этому процессу.
2. **Пересчет динамических приоритетов**
   1. **Windows**

В системе Windows реализовано вытесняющее планирование на основе уровней приоритета, при которой выполняется готовый поток с наивысшим приоритетом. Процессорное время, выделенное на выполнение потока, называется квантом. Если поток с более высоким приоритетом готов к выполнению, текущий поток вытесняется планировщиком, даже если квант текущего потока не истек. В Windows за планирование отвечает совокупность процедур ядра, называемая диспетчером ядра. Диспетчеризация может быть вызвана, если:

* поток готов к выполнению;
* истек квант текущего потока;
* поток завершается или переходит в состояние ожидания;
* изменился приоритет потока;
* изменилась привязка потока к процессору.
  + 1. ***Приоритеты потоков***

В системе предусмотрено 32 уровня приоритетов: уровни реального времени (16–31), динамические уровни (1–15) и системный уровень (0). Уровни приоритета потоков назначаются Windows API и ядром 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).

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);
* **самый низший** — Lowest (–2);
* **уровень простоя** — Idle (–15).

Относительный приоритет — это приращение к базовому приоритету процесса. Соответствие между приоритетами Windows API и ядра системы приведено в таблице.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **реальное время** | **высокий** | **выше обычного** | **обычный** | **ниже обычного** | **простоя** |
| **критичный по времени** | 31 | 15 | 15 | 15 | 15 | 15 |
| **наивысший** | 26 | 15 | 12 | 10 | 8 | 6 |
| **выше обычного** | 25 | 14 | 11 | 9 | 7 | 5 |
| **обычный** | 24 | 13 | 10 | 8 | 6 | 4 |
| **ниже обычного** | 23 | 12 | 9 | 7 | 5 | 3 |
| **самый низший** | 22 | 11 | 8 | 6 | 4 | 2 |
| **уровень простоя** | 16 | 1 | 1 | 1 | 1 | 1 |

Каким бы образом ни формировался приоритет потока, с точки зрения планировщика Windows важно только значение приоритета. Процесс обладает только базовым приоритетом, тогда как поток имеет базовый, который наследуется от приоритета процесса, и текущий приоритет. Операционная система может на короткие интервалы времени повышать приоритеты потоков из динамического диапазона, но никогда не регулирует приоритеты потоков в диапазоне реального времени. Приложения пользователя запускаются, как правило, с базовым приоритетом Normal. Некоторые системные процессы имеют приоритет выше 8, следовательно, это гарантирует, что потоки в этих процессах будут запускаться с более высоким приоритетом. Система динамически повышает приоритет текущего потока в следующих случаях:

* по завершении операции ввода-вывода;
* по окончании ожидания на событии или семафоре исполнительной системы;
* по окончании ожидания потоками активного процесса;
* при пробуждении GUI-потоков из-за операции с окнами;
* если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени.

Динамическое повышение приоритета применяется только к потокам из динамического диапазона (1–15) и, независимо от приращения, приоритет потока не может оказаться выше 15.

* + 1. ***Повышение приоритета по завершении операции ввода-вывода***

По окончании определенных операций ввода-вывода Windows временно повышает приоритет потоков и потоки, ожидающие завершения этих операций, имеют больше шансов немедленно возобновить выполнение и обработать полученные от устройств ввода-вывода данные. Драйвер устройства вводавывода через функцию **𝐼𝑜𝐶𝑜𝑚𝑝𝑙𝑒𝑡𝑒𝑅𝑒𝑞𝑢𝑒𝑠𝑡** указывает на необходимость динамического повышения приоритета после выполнения соответствующего запроса.

В таблицеприведены приращения приоритетов.

|  |  |
| --- | --- |
| **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 приоритета. На рисунке 1 показано, что после повышения приоритета поток в течение одного кванта выполняется с повышенным приоритетом, а затем приоритет снижается на один уровень с каждым последующим квантом. Цикл продолжается до тех пор, пока приоритет не снизится до базового.

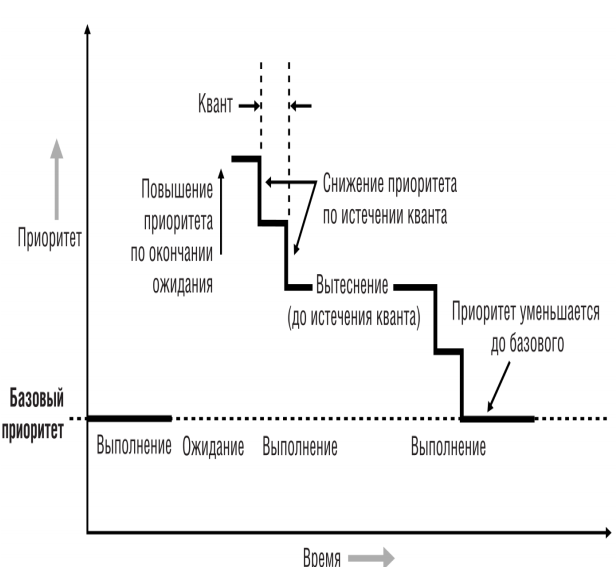


Рисунок 1. Динамическое изменение приоритета

* + 1. ***Повышение приоритета по окончании ожидания на событии или семафоре***

Если ожидание потока на событии системы или «семафоре» успешно завершается из-за вызова **𝑆𝑒𝑡𝐸𝑣𝑒𝑛𝑡**, **𝑃𝑢𝑙𝑠𝑒𝐸𝑣𝑒𝑛𝑡** или **𝑅𝑒𝑙𝑒𝑎𝑠𝑒𝑆𝑒𝑚𝑎𝑝ℎ𝑜𝑟𝑒**, его приоритет повышается на 1. Такая регулировка, как и в случае с окончанием операции ввода-вывода, позволяет равномернее распределить процессорное время — потокам, блокируемым на событиях, процессорное время требуется реже, чем остальным. В данном случае действуют те же правила динамического повышения приоритета. К потокам, пробуждающимся в результате установки события вызовом функций **𝑁𝑡𝑆𝑒𝑡𝐸𝑣𝑒𝑛𝑡𝐵𝑜𝑜𝑠𝑡𝑃𝑟𝑖𝑜𝑟𝑖𝑡𝑦** и **𝐾𝑒𝑆𝑒𝑡𝐸𝑣𝑒𝑛𝑡𝐵𝑜𝑜𝑠𝑡𝑃𝑟𝑖𝑜𝑟𝑖𝑡𝑦**, повышение приоритета применяется особым образом

* + 1. ***Повышение приоритета по окончании ожидания потоками активного процесса***

Если поток в активном процессе завершает ожидание на объекте ядра, функция ядра **𝐾𝑖𝑈𝑛𝑤𝑎𝑖𝑡𝑇ℎ𝑟𝑒𝑎𝑑** повышает его текущий приоритет на величину значения **𝑃𝑠𝑃𝑟𝑖𝑜𝑟𝑖𝑡𝑦𝑆𝑒𝑝𝑎𝑟𝑎𝑡𝑖𝑜𝑛**. **𝑃𝑠𝑃𝑟𝑖𝑜𝑟𝑖𝑡𝑢𝑆𝑒𝑝𝑎𝑟𝑎𝑡𝑖𝑜** — это индекс в таблице квантов, с помощью которой выбираются величины квантов для потоков активных процессов. Какой процесс является в данный момент активным, определяет подсистема управления окнами.

В данном случае приоритет повышается для создания преимуществ интерактивным приложениям по окончании ожидания, в результате чего повышаются шансы на немедленное возобновление потока приложения.

Важной особенностью данного вида динамического повышения приоритета является то, что он поддерживается всеми системами Windows и не может быть отключен даже функцией **𝑆𝑒𝑡𝑇ℎ𝑟𝑒𝑎𝑑𝑃𝑟𝑖𝑜𝑟𝑖𝑡𝑦𝐵𝑜𝑜𝑠𝑡**.

* + 1. ***Повышение приоритета при пробуждении GUI-потоков***

Приоритет потоков окон пользовательского интерфейса повышается на 2 после их пробуждения из-за активности подсистемы управления окнами. Приоритет повышается по той же причине, что и в предыдущем случае, — для увеличения отзывчивости интерактивных приложений.

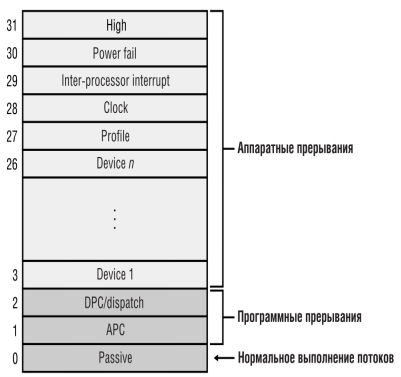
* + 1. ***Повышение приоритета при нехватке процессорного времени***

Раз в секунду диспетчер настройки баланса — системный поток, предназначенный для выполнения функций управления памятью — сканирует очереди готовых потоков и ищет потоки, которые находятся в состоянии готовности в течение примерно 4 секунд. Диспетчер настройки баланса повышает приоритет таких потоков до 15. Причем в Windows 2000 и Windows XP квант потока удваивается относительно кванта процесса, а в Windows Server 2003 квант устанавливается равным 4 единицам. По истечении кванта приоритет потока снижается до исходного уровня. Если потоку все еще не хватило процессорного времени, то после снижения приоритета он возвращается в очередь готовых процессов. Через 4 секунды он может снова получить повышение приоритета.

Чтобы свести к минимуму расход процессорного времени, диспетчер настройки баланса сканирует только 16 готовых потоков за раз, а повышает приоритет не более чем у 10 потоков за раз. Диспетчер настройки баланса не решает всех проблем с приоритетами потоков, однако позволяет потокам, которым не хватает процессорного времени, получить его.

* + 1. ***Уровни запросов прерываний***

Windows использует схему приоритетов прерываний, называемую уровни запросов прерываний (IRQL). Внутри ядра IRQL представляются в виде номеров до 0 до 31 для систем x86. Ядро определяет стандартный набор IRQL для программных прерываний, а HAL связывает IRQL с номерами аппаратных прерываний.



Прерывания обслуживаются в порядке их приоритета. Прерывания с б´ольшим приоритетом вытесняют прерывания с меньшим приоритетом. 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.

* 1. **UNIX**

Классическое ядро UNIX было невытесняемым. Это означает, что если процесс выполняется в режиме ядра, то ядро не заставит этот процесс уступить процессорное время какому-либо более приоритетному процессу. Выполняющийся процесс может освободить процессор в случае своего блокирования в ожидании ресурса, иначе он может быть вытеснен при переходе в режим задачи. Такая реализация ядра позволяет решить множество проблем синхронизации, связанных с доступом нескольких процессов к одним и тем же структурам данных ядра.

Однако современные ядра Linux, начиная с версии 2.5, являются полностью вытесняемыми, так как должны обеспечивать работу процессов реального времени.

* + 1. ***Приоритеты процессов***

Приоритет процесса в UNIX задается числом в диапазоне от 0 до 127, причем чем меньше значение, тем выше приоритет. Приоритеты 0–49 зарезервированы ядром операционной системы, прикладные процессы могут обладать приоритетом в диапазоне от 50 до 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 поле 𝑝\_𝑝𝑟𝑖 для принятия решения о том, какой процесс отправить на выполнение. Значения 𝑝\_𝑝𝑟𝑖 и 𝑝\_𝑢𝑠𝑟𝑝𝑟𝑖 идентичны, когда процесс находится в режиме задачи. Когда процесс просыпается после блокировки в системном вызове, его приоритет временно повышается. Планировщик использует 𝑝\_𝑢𝑠𝑟𝑝𝑟𝑖 для хранения приоритета, который будет назначен процессу при переходе из режима ядра в режим задачи, а 𝑝\_𝑝𝑟𝑖 — для хранения временного приоритета для выполнения в режиме ядра. Ядро связывает приоритет сна (0–49) с событие или ожидаемым ресурсом, из-за которого процесс может быть заблокирован. Когда блокированный процесс просыпается, ядро устанавливает 𝑝\_𝑝𝑟𝑖, равное приоритету сна события или ресурса, на котором он был заблокирован, следовательно, такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи.

|  |  |  |
| --- | --- | --- |
| Событие | Приоритет 4.3BSD UNIX | Приоритет SCO UNIX |
| Ожидание загрузки в память страницы | 0 | 95 |
| Ожидание индексного дескриптора | 10 | 88 |
| Ожидание ввода–вывода | 20 | 81 |
| Ожидание буфера | 30 | 80 |
| Ожидание терминального ввода | 30 | 75 |
| Ожидание терминального вывода | 30 | 74 |
| Ожидание завершения выполнения | 30 | 73 |
| Ожидание события | 40 | 66 |

В таблице приведены значения приоритетов сна для систем 4.3BSD UNIX и SCO UNIX (OpenServer 5.0). Такой подход позволяет системным вызовам быстрее завершать свою работу. По завершении процессом системного вызова его приоритет сбрасывается в значение текущего приоритета в режиме задачи. Если при этом приоритет окажется ниже, чем приоритет другого запущенного процесса, ядро произведет переключение контекста. Приоритет в режиме задачи зависит от «любезности» и последней измеренной величины использования процессора. Степень любезности — это число в диапазоне от 0 до 39 со значение 20 по умолчанию. Степень любезности называется так потому, что одни пользователи могут быть поставлены в более выгодные условия другими пользователями посредством увеличения кем-либо из последних значения уровня любезности для своих менее важных процессов. Системы разделения времени стараются выделить процессорное время таким образом, чтобы все процессы системы получили его в примерно равных количествах, что требует слежения за использованием процессора. Поле 𝑝\_𝑐𝑝𝑢 содержит величину последнего измерения использования процессора процессом. При создании процесса это поле инициализируется нулем. На каждом тике обработчик таймера увеличивает 𝑝\_𝑐𝑝𝑢 на единицу для текущего процесса, вплоть до максимального значения — 127. Каждую секунду ядро вызывает процедуру 𝑠𝑐ℎ𝑒𝑑𝑐𝑝𝑢, которая уменьшает значение 𝑝\_𝑐𝑝𝑢 каждого процесса исходя из фактора «полураспада».

В 4.3 BSD для расчета применяется формула:

где 𝑙𝑜𝑎𝑑\_𝑎𝑣𝑒𝑟𝑎𝑔𝑒 — это среднее количество процессов в состоянии готовности за последнюю секунду

Кроме того, процедура 𝑠𝑐ℎ𝑒𝑑𝑐𝑝𝑢 также пересчитывает приоритеты режима задачи всех процессов по формуле:

где 𝑃 𝑈𝑆𝐸𝑅 — базовый приоритет в режиме задачи, равный 50.

Таким образом, если процесс до вытеснения другим процессом использовал большое количество процессорного времени, его 𝑝\_𝑐𝑝𝑢 будет увеличен, что приведен к увеличению значения 𝑝\_𝑢𝑠𝑟𝑝𝑟𝑖, и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем меньше его 𝑝\_𝑐𝑝𝑢. Это позволяет предотвратить зависания низкоприоритетных процессов. Если процесс б´ольшую часть времени выполнения тратит на ожидание ввода-вывода, то он остается с высоким приоритетом. В системах разделения времени фактор использования процессора обеспечивает справедливость при планировании процессов. Фактор полураспада обеспечивает экспоненциально взвешанное среднее значение использования процессора в течение функционирования процесса. Формула, применяемая в SVR3 имеет недостаток: вычисляя простое экспоненциальное среднее, она способствует росту приоритетов при увеличении загрузки системы.

1. **Вывод**

Операционные системы UNIX и Windows являются системами разделения времени с вытеснением. В связи с этим обработчики прерываний от системных таймеров в них выполняют схожие функции:

* инкремент счетчика системного времени;
* декремент кванта;
* добавление функций планировщика в очередь отложенных вызовов;
* декремент счетчиков времени, оставшегося до выполнения отложенных вызовов;
* отправка отложенных вызовов на выполнение.

Декремент кванта является основной функцией обработчика прерывания от системного таймера. Классическое ядро UNIX было строго невытесняемым. Ядра Linux, начиная с версии 2.5, и ядра операционных систем Windows являются полностью вытесняемыми для обеспечения работы процессов реального времени.