|  |  |
| --- | --- |
| 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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 1(часть 2) |

**Название:**

Функции обработчика прерывания системного таймера. Пересчёт динамических приоритетов

**Дисциплина:** Операционные системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | Н.А. Гарасев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Н. Ю. Рязанова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

# Функции системных таймеров в защищенном режиме

Рассмотрим функции системных таймеров в зависимости от операционной системы.

# Linux, Unix

# По тику:

* + 1. инкремент часов и других таймеров системы;
    2. декремент кванта текущего потока;
    3. декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов.

# По главному тику:

* + 1. декремент времени, оставшегося до отправления одного из сигналов: **SIGALARM** (сигнал по истечении времени, предварительно заданного функцией alarm()); **SIGPROF** (сигнал, посылаемый процессу по истечении времени, заданном в таймере профилирования); SIGVTALARM (сигнал, посылаемый процессу по истечении времени, заданного в «виртуальном» таймере).

# По кванту:

* + 1. посылка текущему процессу сигнала SIGXCPU, если тот превысил выделенный ему квант использования процессора. По получению сигнала обработчик сигнала прерывает выполнение процесса.

# Windows

# По тику:

* + 1. инкремент счетчика системного времени;
    2. декремент кванта текущего потока;
    3. декремент счетчика отложенных задач.

# По главному тику:

* + 1. объекта «событие», которое ожидает диспетчер настройки баланса; диспетчер настройки баланса по событию от таймера сканирует очередь готовых процессов и повышает приоритет процессов.

# По кванту:

* + 1. инициация диспетчеризации потоков (добавление соответствующего объекта в очередь DPC)

# Пересчет динамических приоритетов

И в Unix и в Windows могут пересчитываться приоритеты только пользовательских процессов.

# Linux, Unix

В традиционных системах **Unix** ядро является строго *невытесняющим*, однако в современных системах **Unix** ядро является *вытесняющим* – процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Ядро было сделано вытесняющим для того, чтобы система могла обслуживать процессы реального времени, такие как аудио и видео.

Планирование процессов в **Unix** основывается на *приоритете* *процесса*. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритет, в свою очередь, не является фиксированным и динамически изменяется системой в зависимости от использования вычислительных ресурсов, времени ожидания запуска и имеет наивысший приоритет. Планировщик приостановит выполнение текущего процесса (с более низким приоритетом), даже если еще не выработал свой временной квант.

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**.

Основная стратегия обслуживания, применяемая в **Unix**-системах, — это равенство в обслуживании и обеспечение приемлемого времени реакции системы. Реализуется эта стратегия за счет дисциплины диспетчеризации RR с несколькими очередями и механизма динамических приоритетов. Во-первых, в вычислении участвуют значения двух полей дескриптора процесса — *p\_nice* и *p\_cpu*. Первое из них назначается пользователем явно или формируется по умолчанию с помощью системы программирования. Второе поле формируется диспетчером задач (планировщиком разделения времени) и называется системной составляющей или текущим приоритетом. Другими словами, каждый процесс имеет два атрибута приоритета. С учетом этого приоритета и распределяется между исполняющимися задачами процессорное время: текущий приоритет, на основании которого происходит планирование, и заказанный относительный приоритет (называемый nice number, или просто nice).

Структура *proc* содержит следующие поля, относящиеся к приоритетам:

* *p\_рri* - 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.

На каждом тике обработчик таймера увеличивает *p\_cpu*на единицу для текущего процесса до максимума.

Каждую секунду ядро системы вызывает процедуру *shedcpu(),* которая запускается через отложенный вызов и уменьшает значение *p\_cpu*каждого процесса на основе фактора полураспада:

В данном случае *load\_average* – среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду. Фактор полураспада обеспечивает экспоненциально взвешенное среднее значение использования процессора в течение всего времени его функционирования.

Процедура *shedcpu()*пересчитывает приоритеты для режима задачи всех процессов по формуле

,

Где *puser* **–** базовый приоритет 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 *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.

Таблица 1. Системные приоритеты сна

|  |  |  |
| --- | --- | --- |
| Событие | Приоритет (4.3BSD Unix) | Приоритет (SCO Unix) |
| Ожидание загрузки сегмента/страницы в память | 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 | 40 | 66 |

Процессу, ожидающему недоступный в данный момент ресурс, система определяет значение *приоритета сна*, выбираемое ядром из диапазона системных приоритетов на основе связи с событием, вызвавшим это состояние.

# Windows

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

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом. Windows осуществляет планировку потоков по вытесняющему принципу. Такой подход имеет смысл, если учесть, что процессы не запускаются, а только предоставляют ресурсы и контекст, в котором запускаются потоки.

В Windows код, отвечающий за планирование, реализован в ядре. В Windows нет единого модуля или процедуры под названием «планировщик». Совокупность процедур, выполняющие эти обязанности, называются диспетчером ядра.

Диспетчеризации потоков могут потребовать следующие события:

* Поток становится готовым к выполнению. Например, поток был только что создан или только что вышел из состояния ожидания;
* Поток выходит из состояния выполнения из-за окончания его кванта времени, его работа завершается, либо переходит в состояние ожидания;
* Изменяется привязка потока к процессору, из-за чего поток не может больше выполняться на текущем процессоре.

В Windows используется 32 уровня приоритета: целое число от 0 до 31, где 31 – наивысший приоритет, из них:

* от 16 до 31 – уровни реального времени
* от 0 до 15 – динамические уровни, уровень 0 зарезервирован для потока обнуления страниц

Уровни приоритета потоков назначаются от Windows API и от ядра Windows.

Сначала Windows API систематизирует процессы по классу приоритета, который им присваивается при создании:

* Реального времени — Real-time (4)
* Высокий — High (3)
* Выше обычного — Above Normal (7)
* Обычный — Normal (2)
* Ниже обычного — Below Normal (5)
* Простоя — Idle (1)

Затем назначается относительный приоритет отдельных потоков внутри этих процессов. Здесь номера представляют изменение приоритета, применяющееся к базовому приоритету процесса:

* 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)

Отображение приоритетов ядра Windows на Windows API показано в таблице 2.

Таблица 2. Отображение приоритетов ядра Windows на Windows API

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Класс приоритета/ Относительный приоритет | Realtime | High | Above | 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 (- насыщение) | 16 | 1 | 1 | 1 | 1 | 1 |

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 повышен планировщиком вследствие следующих причин:

* Повышение вследствие событий планировщика или диспетчера (сокращение задержек)
* Повышения приоритета, связанные с завершением ожидания
* Повышение приоритета владельца блокировки
* Повышение вследствие завершения ввода-вывода (сокращение задержек)

Рекомендуемые приращения приоритета:

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

* Повышение при ожидании ресурсов исполняющей системы
* Повышение приоритета потоков первого плана после ожидания
* Повышение приоритета после пробуждения GUI-потока
* Повышения приоритета, связанные с перезагруженностью центрального процессора.
* Повышение приоритетов для мультимедийных приложений и игр

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

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

1. инициализируют отложенные действия, относящиеся к работе планировщика, такие как пересчет приоритетов;

2. выполняют декремент счетчиков времени;

3. выполняют декремент кванта.

Пересчет приоритетов пользовательских процессов выполняется с целью исключения бесконечного откладывания и увеличения отзывчивости системы на действия пользователя.