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 дисциплине «Операционные системы»

Тема Прерывания таймера в Windows и UNIX

Студент Фролов Е. А.

Группа ИУ7-55Б

Преподаватель Рязанова Н. Ю.

Москва — 2021 г.

Функции обработчика прерывания от системного таймера в защищённом режиме

1.1 Windows-системы

По тику:

- инкремент счётчика системного времени;
- декремент остатка кванта текущего потока;
- декремент счетчиков времени отложенных задач;
- в случае, если активен механизм профилирования ядра, инициализирует отложенный вызов обработчика ловушки профилирования ядра путём постановки объекта в очередь **DPC**: обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

По главному тику:

- инициализация диспетчера настройки баланса путем освобождения объекта "событие"каждую секунду.

По кванту:

- инициализация диспетчеризации потоков посредством добавления соответствующего объекта **DPC** в очередь.

1.2 UNIX-системы

По тику:

- инкремент счетчика тиков аппаратного таймера;
- инкрементт 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;
- инкремент счетчика использования процессора текущим процессом;
- декремент кванта текущего потока.

По главному тику:

- добавляет в очередь отложенных вызовов функций, относящиеся к работе планировщика, такие как пересчет приоритетов;
- пробуждение системных процессов, таких, как **swapper** и **pagedaemon** (процедура **wakeup** перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»)
- декремент счетчика 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:

- при превышении текущим процессом выделенного кванта, отправка сигнала **SIGXCPU** этому процессу.

Пересчёт динамических приоритетов

2.1 UNIX-системы

Классическое ядро UNIX является строго 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.

В современных системах UNIX/Linux ядро является вытесняемым.

Приоритет процесса задается любым целым числом от 0 до 127. Чем меньше число , тем выше приоритет.

- 0 - 49 - зарезервированы для ядра;
- 50 - 127 - приоритеты прикладных процессов.

В первую очередь выполняются процессы с большим приоритетом, а процессы с одинаковыми приоритетами выполняются в течении кванта времени циклически друг за другом.

2.1.1 Приоритеты процессов

Приоритеты ядра являются фиксированными величинами, а приоритеты прикладных задач могут изменяться во времени в зависимости от следующих факторов:

Дескриптор процесса **proc** содержит следующие поля, которые относятся к приоритету процесса:

- **p_pri** - текущий приоритет планирования;
- **p_usrpri** – приоритет режима задачи;
- **p_cpu** – результат последнего измерения использования процессора; (процессом);
- **p_nice** – фактор "любезности устанавливаемый пользователем.

Для решения, какой процесс направить на выполнение, планировщик использует **p_pri**. В режиме задачи, у процесса значения **p_pri** и **p_usrpri** идентичны. В режиме ядра, планировщиком значение текущего приоритета **p_pri** может быть повышено. Тогда, **p_usrpri** будет использоваться для хранения приоритета, который будет назначен процессу при возврате в режим задачи.

Фактор любезности — целое число в диапазоне от 0 до 39 со значением 20 по умолчанию.

При увеличении фактора любезности, уменьшается приоритет процесса. Фактор любезности процесса может быть изменен суперпользователем с помощью системного вызова nice. При создании процесса, поле **p_cpu** инициализируется нулем. На каждом тике обработчик таймера увеличивает поле

p_cpu текущего процесса на единицу, до максимального значения, равного 127.

Ядро связывает приоритет сна (0—49) с событием 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.

На рисунке 2.1 приведены значения приоритетов сна для систем 4.3BSD UNIX и SCO UNIX. 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	Значение	Описание
PSWP	0	Свопинг
PSWP + 1	1	Страничный демон
PSWP + 1/2/4	1/2/4	Другие действия по обработке памяти
PINOD	10	Ожидание освобождения inode
PRIBIO	20	Ожидание дискового ввода-вывода
PRIBIO + 1	21	Ожидание освобождения буфера
PZERO	25	Базовый приоритет
TTIPRI	28	Ожидание ввода с терминала
TTOPRI	29	Ожидание вывода с терминала

Рисунок 2.1 – Системные приоритеты сна

Каждую секунду, обработчик прерывания инициализирует отложенный вызов процедуры schedcpu(), которая уменьшает значение p_cpu каждого

процесса исходя из фактора "полураспада", который рассчитывается по формуле 2.1, где *load_ average* - это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

$$decay = \frac{2 \cdot load\_average}{2 \cdot load\_average + 1} \tag{2.1}$$

Процедура schedcpu() пересчитывает приоритеты для режима задачи всех процессов по формуле 2.2, где *PUSER* - базовый приоритет в режиме задачи, равный 50.

$$p\_usrpri = PUSER + \frac{p\_cpu}{2} + 2 \cdot p\_nice \tag{2.2}$$

В результате, если процесс в последний раз использовал большое количество процессорного времени, его **p_cpu** будет увеличен. Это приведет к росту **p_usrpri** и к понижению приоритета. Такая схема отдает предпочтение процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений

2.2 Windows-системы

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.

2.2.1 Приоритеты потоков

Windows использует 32 уровня приоритета, от 0 до 31. Эти значения разбиваются на части следующим образом:

- шестнадцать уровней реального времени (от 16 до 31);
- шестнадцать динамических уровней (от 0 до 15), из которых уровень 0 зарезервирован для потока обнуления страниц.

Уровни приоритета потоков назначаются исходя из двух разных позиций: одной от Windows API и другой от ядра Windows. Сначала Windows API сортирует процессы по классу приоритета, который им присваивается при 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 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);
- 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. Соответствие между приоритетами Windows API и ядра системы приведено на рисунке 2.2.

Текущий приоритет потока в динамическом диапазоне — от 1 до 15 — может быть повышен планировщиком вследствие следующих причин:

- повышение вследствие событие планировщика или диспетчера(сокращение задержек);

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

Рисунок 2.2 – Соответствие между приоритетами Windows API и ядра Windows

- повышение приоритета владельца блокировки;
- повышение приоритета после завершения ввода/вывода (сокращение задержек) (рисунок 2.3);
- повышение приоритета вследствие ввода из пользовательского интерфейса(сокращение задержек и времени отклика);
- повышение приоритета вследствие длительного ожидания ресурса исполняющей системы(предотвращение зависания);
- повышение вследствие ожидания объекта ядра;
- повышение приоритета в случае, когда готовый к выполнению поток не был запущен в течение длительного времени (предотвращение зависания и смены приоритетов);
- повышение приоритета проигрывания мультимедиа службой планировщика `MMCSS`.

Устройство	Повышение приоритета
Жесткий диск, привод компакт-дисков, параллельный порт, видеоустройство	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковое устройство	8

Рисунок 2.3 – Рекомендуемые значения повышения приоритета

2.3 MMCSS.

Потоки, на которых выполняются различные мультимедийные приложения, должны выполняться с минимальными задержками. В Windows эта задача решается путем повышения приоритетов таких потоков драйвером **MMCSS** – MultiMedia Class Scheduler Service. Приложения, которые реализуют воспроизведение мультимедиа, указывают драйверу **MMCSS** задачу из списка:

- аудио;
- захват;
- распределение;
- игры;
- проигрывание;
- аудио профессионального качества;
- задачи администратора многооконного режима.

В свою очередь, 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, которое является первичным фактором, определяющим приоритет потоков, зарегистрированных с MMCSS. На рисунке 2.4 показаны различные категории планирования.

Механизм, положенный в основу MMCSS, повышает приоритет потоков внутри зарегистрированного процесса до уровня, соответствующего их категории планирования.

Категория	**Приоритет**	**Описание**
High (Высокая)	23–26	Потоки профессионального аудио (Pro Audio), запущенные с приоритетом выше, чем у других потоков на системе, за исключением критических системных потоков
Medium (Средняя)	16–22	Потоки, являющиеся частью приложений первого плана, например Windows Media Player
Low (Низкая)	8–15	Все остальные потоки, не являющиеся частью предыдущих категорий
Exhausted (Исчерпавших потоков)	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

Рисунок 2.4 – Категории планирования

Затем он снижает категорию этих потоков до Exhausted, чтобы другие, не относящиеся к мультимедийным приложениям потоки, также могли получить ресурс.

Вывод

Windows и Unix обработчик прерывания системного таймера выполняет очень похожие функции т.к. обе эти системы являются системами разделения времени:

- инициализируют отложенные действия (такие как пересчет приоритетов);
- выполняют декремент счетчиков времени:
  - часов;
  - таймеров;
  - будильников реального времени;
  - счетчиков времени отложенных действий.
- уменьшает квант процессорного времени, выделенного процессу.

Декремент кванта является основной функцией обработчика прерывания от системного таймера.

Системы планирования в этих ОС различаются: Windows – полностью вытесняющая, классический Unix – строго невытесняющая, при этом, ядро Linux, начиная с версии 2.5, является полностью вытесняющим, для обеспечения работы процессов реального времени.

Приоритет пользовательского процесса в ОС Unix/Linux, в зависимости от фактора любезности, p_cpu и базового приоритета может динамически пересчитываться.

В Windows, процессу назначается базовый приоритет. Процессу потока назначается приоритет, относительно базового.

Литература

[1] Вахалия. UNIX изнутри..

[2] Соломон, Руссинович. Внутреннее устройство Windows.