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 Прерывания таймера в Windows и Unix

Студент Хамзина Р. Р.

Группа ИУ7-53Б

Преподаватель Рязанова Н. Ю.

Москва — 2021 г.

Содержание

1 Функции обработчика прерывания от системного таймера в защищенном режиме

1.1 ОС семейства Unix

1.1.1 По тику

Период времени между двумя последующими прерываниями таймера называется *тиком*. В задачи обработчика прерывания от системного таймера по тику входит:

- инкремент счетчика тиков аппаратного таймера;
- инкремент часов и других таймеров системы;
- декремент кванта;
- декремент счетчика времени до отправления на выполнение отложенного вызова;
- обновление статистики использования процессора текущим процессом : инкремент поля p_cpu структуры proc.

1.1.2 По главному тику

Период времени, который равен n тикам таймера (число n зависит от конкретного варианта системы), называют *главным тиком*. В задачи обработчика прерывания от системного таймера по главному тику входит:

- регистрация отложенных вызовов функций, относящихся к работе планировщика;

- пробуждение системных процессов - распознавание отложенного вызова процедуры *wakeup*, которая перемещает дескрипторы процессов из списка "спящих" в очередь "готовых к выполнению";

- декремент счетчика времени, которое осталось до отправления одного из следующих сигналов:

  – *SIGALRM* - сигнал, посылемый процессу по истечении времени, которое предварительно задано функцией *alarm()*;

  – *SIGPROF* - сигнал, посылаемый процессу по истечении времени, которое задано в таймере профилирования;

  – *SIGVTALRM* - сигнал, посылаемый процессу по истечении времени, которое задано в "виртуальном" таймере.

1.1.3 По кванту

Временной интервал, в течение которого процесс может использовать процессор до вытеснения другим процессом, называют *квантом времени*. Обработчик прерывания от системного таймера по кванту посылает сигнал *SIGXCPU* текущему процессу, если он израсходовал выделенный ему квант времени.

1.2 ОС семейства Windows

1.2.1 По тику

В задачи обработчика прерывания от системного таймера по тику входит:

- инкремент счетчика системного времени;

- декремент кванта текущего потока на величину, равную количеству тактов процессора, произошедших за тик;

- декремент счетчика времени отложенных задач;
- инициализация отложенного вызова обработчика ловушки профилирования ядра путем постановки объекта в очередь DPC, если активен механизм профилирования ядра.

1.2.2 По главному тику

Обработчик прерывания от системного таймера по кванту освобождает объект "событие", который ожидает диспетчер настройки баланса.

1.2.3 По кванту

Обработчик прерывания от системного таймера по кванту инициализирует диспетчеризацию потоков путем постановки соответствующего объекта в очередь DPC.

2 Пересчет динамических приоритетов

В ОС семейства Unix и в ОС семейства Unix динамически пересчитываться могут только приоритеты пользовательских процессов.

2.1 ОС семейства Unix

Очередь процессов, готовых к выполнению, формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования.

Приоритет задается целым числом из диапазона от 0 до 127. Чем меньше число, тем выше приоритет процесса. Приоритеты ядра находятся в диапазоне от 0 до 49 и являются фиксированными величинами, а приоритеты прикладных задач - в диапазоне от 50 до 127.

Процессы с большим приоритетом выполняются в первую очередь, процессы с одинаковыми приоритетами выполняются в течении кванта времени циклически друг за другом. Когда в очередь поступает процесс с более высоким приоритетом, планировщик предоставляет ресурс более приоритетному процессу, вытесняя текущий.

Традиционное ядро Unix - строго невытесняемое, то есть, ядро не заставит процесс, выполняющийся в режиме ядра, уступить процессор какому-либо высокоприоритетному процессу. В современных системах Unix ядро - вытесняющее: процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Ядро было сделано вытесняющим, чтобы система могла обслуживать процессы реального времени (видео или аудио).

Приоритеты прикладных задач могут изменяться во времени в зависимости от двух факторов:

- фактор любезности. Чем меньше его значение, тем выше приоритет процесса. Суперпользователь может повысить приоритет с помощью системного вызова *nice*;
- последней измеренной величины использования процессора.

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

- *_pri* – текущий приоритет планирования;
- *p_usrpri* – приоритет процесса в режиме задачи;
- *p_cpu* – результат последнего измерения степени загруженности процессора процессом;
- *p_nice* – фактор любезности.

У процесса, который 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 (*p_usrpri* будет использоваться для хранения приоритета, который будет назначен при возврате в режим задачи).

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. Когда процесс "просыпается" после блокирования в системном вызове, ядро устанавливает в поле *p_pri* приоритет сна – значение приоритета из диапазона от 0 до 49, зависящее от события или ресурса по которому произошла блокировка. Значения приоритетов сна для событий в системе *4.3BSD* описаны в таблице 2.1.

Таблица 2.1 – Приоритеты сна в 4.3BSD

Приоритет	Значение	Описание
PSWP	0	Свопинг
PSWP + 1	1	Страничный демон
PSWP + 1/2/4	1/2/4	Другие действия при 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
TTORPI	29	Ожидание вывода с терминала

Системные приоритеты сна для *4.3BSD UNIX* и *SCO UNIX* приведены в таблице 2.2.

Таблица 2.2 – Системные приоритеты сна

Событие	Приоритет 4.3BSD UNIX	Приоритет SCO 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 нулем. На каждом тике обработчик таймера увеличивает поле *p_cpu* текущего процесса на единицу, до максимального значения, равного 127. Каждую секунду, обработчик прерывания инициализирует отложенный вызов процедуры *schedcpu()*, которая уменьшает значение *p_cpu* каждого процесса исходя из фактора "полураспада".

В системе *4.3BSD* для расчёта фактора полураспада применяется формула (2.1).

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

где $load\_average$ - это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

Процедура *schedcpu()* пересчитывает приоритеты для режима задачи всех процессов по формуле (2.2).

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

где $PUSER$ - базовый приоритет в режиме задачи, равный 50.

p_cpu будет увеличен, если процесс в последний раз использовал большое количество процессорного времени. Это приведет к росту значения *p_usrpri*, из чего следует понижение приоритета. Чем дольше процесс про-

стаивает в очереди на исполнение, тем больше фактор полураспада уменьшает его p_cpu, что приводит к повышению его приоритета. Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов в ОС. Ее применение предпочтительнее процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

2.2 ОС семейства Windows

В Windows процессу при создании назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается относительный приоритет.

Планирование осуществляется только на основании приоритетов потоков, готовых к выполнению: если поток с более высоким приоритетом становится готовым к выполнению, поток с более низким приоритетом вытесняется планировщиком. По истечению кванта времени текущего потока, ресурс передается самому приоритетному потоку в очереди готовых к выполнению.

В Windows используется 32 уровня приоритета - целое число от 0 до 31 (наивысший):

- от 0 до 15 - изменяющиеся уровни. Уровень 0 зарезервирован для потока обнуления страниц;
- от 16 до 31 - уровни реального времени.

Уровни приоритета потоков назначаются в зависимости от двух разных позиций: Windows API и 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) - (6);

- обычный (Normal) - (2);
- ниже обычного (Below Normal) - (5);
- уровень простоя (Idle) - (1).

Затем назначается относительный приоритет отдельных 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);
- самый низший (Lowest) - (–2);
- уровень простоя (Idle) - (–15)

Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал.

В таблице 2.3 показано соответствие между приоритетами Windows API и ядра системы.

Таблица 2.3 – Соответствие между приоритетами Windows API и ядра Windows

Класс приоритета	Real-time	High	Above	Normal	Below Normal	Idle
Time Critical	31	15	15	15	15	15
Highest	26	15	12	10	8	6
Above Normal	25	14	11	9	7	5
Normal	24	13	10	8	6	4
Below Normal	23	12	9	7	5	3
Lowest	22	11	8	6	4	2
Idle	16	1	1	1	1	1

В Windows также включен *диспетчер настройки баланса*. Он раз в секунду сканирует очередь готовых потоков. Если обнаружены потоки, ожидающие выполнения более 4 секунд, диспетчер настройки баланса повышает их приоритет до 15. Когда истекает квант, приоритет потока снижается до базового приоритета. Если поток не был завершен за квант времени или был вытеснен потоком с более высоким приоритетом, то после снижения приоритета поток возвращается в очередь готовых потоков.

Для того, чтобы минимизировать расход процессорного времени, диспетчер настройки баланса сканирует только 16 потоков и повышает приоритет не более чем у 10 потоков за одни проход. Когда обнаружится 10 потоков, приоритет которых следует повысить, диспетчер настройки баланса прекращает сканирование. При следующем проходе возобновляет сканирование с того места, где оно было прервано.

Планировщик может повысить текущий приоритет потока в динамическом диапазоне (от 1 до 15) вследствие следующих причин:

- повышение вследствие событие планировщика или диспетчера;
- повышение приоритета владельца блокировки;
- повышение приоритета после завершения ввода/вывода (таблица 2.4);

Таблица 2.4 – Рекомендуемые значения повышения 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 приоритета вследствие ввода из пользовательского интерфейса;
- повышение приоритета вследствие длительного ожидания ресурса исполняющей системы;
- повышение вследствие ожидания объекта ядра;

- повышение приоритета в случае, когда готовый к выполнению поток не был запущен в течение длительного времени;

- повышение приоритета проигрывания для мультимедийных приложений. Для того, чтобы мультимедийные потоки могли выполняться с минимальными задержками, функции *MMCSS* (*MultiMedia Class Scheduler Service*) временно повышают приоритет потоков до уровня, соответствующего их категориям планирования (таблица 2.5). Для того, чтобы другие потоки могли получить ресурс, приоритет снижается до уровня, соответствующего категории *Exhausted*.

Таблица 2.5 – 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 продолжиться, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета

Вывод

Обработчик прерывания от системного таймера для ОС семейства Windows и для ОС семейства Unix выполняет следующие общие функции:

- декремент счетчиков времени (часов, таймеров, счетчиков времени отложенных действий, будильников реального времени);
- декремент кванта;
- инициализация отложенных действий, которые относятся к работе планировщика.

ОС Unix и ОС Windows - системы разделения времени с динамическими приоритетами и вытеснением.

Пересчет динамических приоритетов в ОС Unix происходит следующим образом. Приоритет процесса характеризуется текущим приоритетом и приоритетом процесса в режиме задачи. Приоритеты ядра - фиксированные величины, а приоритеты пользовательского процесса (процесса в режиме задачи) могут быть динамически пересчитаны исходя из фактора любезности и величины использования процессора.

Пересчет динамических приоритетов в ОС Windows можно описать следующим образом. Приоритет, называемый базовым, назначается процессу при его создании. Приоритеты потоков определяются относительно приоритета процесса, в котором они создаются. Приоритет потока пользовательского процесса может быть пересчитан динамически.