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)

Студент Топорков Павел
Группа ИУ7-53Б
Дисциплина Операционные системы

Преподаватель: _____ Рязанова Н. Ю.
подпись, дата Фамилия, И.О.

Оценка _____

Москва — 2021 г.

Оглавление

1 Функции обработчика прерываний от системного таймера

1.1 Unix

По тику:

- Инкремент часов и других таймеров системы
- Инкремент счетчика тиков аппаратного таймера
- Инкремент поля p_cpu дескриптора текущего процесса
- Декремент счетчика времени до отправления на выполнение отложенных вызовов
- Декремент кванта текущего потока

По главному тику:

- Инициализация отложенных вызовов функций, которые относятся к работе планировщика(пересчёт приоритетов). Инициализация отложенного вызова заключается в посылке соответствующего сигнала или в изменении состояния процесса с S на R.
- Пробуждение системных процессов, таких, как swapper и pagedaemon (процедура wakeup перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»)
- Декремент счетчиков времени, оставшегося до отправления сигналов тревоги:
  - SIGALRM - будильники (например, функция alarm());
  - SIGPROF - посылается процессу по истечении времени, заданного в таймере профилирования (данный таймер уменьшается только когда процесс выполняется в режиме ядра или задачи) - измеряет время выполнения самого процесса и время, проведенное в ожидании завершения системных вызовов;

– SIGVTALRM - в отличие от SIGALRM, который измеряет реальное время, и SIGPROF, который измеряет время выполнения самого процесса и время, проведенное в ожидании завершения системных вызовов, SIGVTALRM измеряет только время выполнения процесса.

По кванту:

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

1.2 Windows

По тику:

- инкремент системного времени
- декремент счетчиков отложенных задач
- декремент остатка кванта текущего потока
- активизация обработчика ловушки профилирования ядра (добавление процесса в очередь DPC)

По главному тику:

- инициализация диспетчера настройки баланса (путем освобождения объекта «событие», на котором он ожидает).

По кванту:

- инициация диспетчеризации потоков (посредством добавления соответствующего объекта DPC в очередь).

2 Пересчет динамических приоритетов

2.1 Unix

Очередь готовых к выполнению процессов формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования: в первую очередь выполняются процессы с наивысшим приоритетом, а процессы с одинаковыми приоритетами выполняются в течении кванта времени циклически друг за другом. Если в очередь готовых к выполнению процессов поступает процесс, который имеет более высокий приоритет, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному. Стоит отметить, что динамические приоритеты могут быть только у пользовательских процессов - они изменяются системой в зависимости от текущего состояния процесса, времени ожидания запуска, использования процессом вычислительных ресурсов.

Приоритет процесса задается любым целым числом от 0 до 127 (приоритеты от 0 до 49 – зарезервировано для ядра, а прикладные процессы обладают диапазоном от 50 до 127). Чем меньше число, тем выше приоритет процесса. Структура proc содержит следующие поля, относящиеся к приоритетам:

p_pri	Текущий приоритет планирования	Используется для хранения временного приоритета для выполнения в режиме ядра
p_uspri	Приоритет режима задачи	Используется для хранения приоритета, который будет назначен процессу при возврате в режим задачи
p_cpu	Результат последнего измерения использования процессора	Содержит величину результата последнего сделанного измерения использования процессора процессом. Инициализируется нулем.
p_nice	Фактор nice, устанавливаемый пользователем	Увеличение значения приводит к уменьшению приоритета.

Данные поля могут изменять свои значения в следующих случаях:

- p_pri равно p_usrpri, когда процесс находится в режиме задачи;
- когда процесс просыпается после блокирования в системном вы- зове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра;
- когда заблокированный процесс просыпается, ядро устанавли- вает значение его p_pri, равное приоритету сна события или ресурса (в диапазоне 0-49);

  Системные приоритеты сна представлены на рисунке 2.1 ниже.
- когда процесс завершил выполнение системного вызова и на- ходится в состоянии возврата в режим задачи, его приоритет сбрасывается обратно значение текущего приоритета в режиме задачи;
- на каждом тике обработчик таймера инкрементирует p_cpu для текущего процесса до максимального значения (127);
- каждую секунду ядро системы вызывает процедуру schedcpu() (запускаемую через отложенный вызов), которая уменьшает зна- чение p_cpu каждого процесса исходя из фактора «полураспа- да» (decay factor); в системе SVR3 используется фиксированное значение этого

фактора, равное 1/2, а в 4.3 BSD для расчета фактора применяется следующая формула:

$$decay = \frac{2 * load\_average}{2 * load\_average + 1},$$

где load_average - это среднее количество процессов, находя- щихся в состоянии готовности к выполнению за последнюю секунду.

Таблица 3.3. 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

Рисунок 2.1: Системные приоритеты сна

Процедура schedcpu() также пересчитывает приоритеты для режима задачи всех процессов по формуле:

$$p\_usercpu = PUSER + \frac{p\_cpu}{4} + 2 * p\_nice,$$

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

В результате, если процесс в последний раз использовал большое количество процессорного времени, его p_cpu 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 p_cpu, что приводит к повышению его приоритета.

2.2 Windows

Ядро Windows создает системный поток, называемый диспетчером набора балансировки (balance set manager); он активизируется один раз в секунду для инициирования различных событий, связанных с планированием и управлением памятью.

Всего в ОС Windows 32 уровня запроса прерывания (от 0 до 31). Прерывания обслуживаются в порядке их приоритета. Эти значения разбиваются на части следующим образом:

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

Уровни приоритета потоков назначаются Windows API и ядром Windows. Сначала WinAPI систематизирует процессы по классу приоритета (присваивается при создании), затем назначается относительный приоритет отдельных потоков внутри этих процессов.

Поэтому в Windows API каждый поток имеет базовый приоритет, являющийся функцией класса приоритета процесса и его относительного приоритета процесса. В ядре класс приоритета процесса преобразуется в базовый приоритет путем использования процедуры PspPriorityTable и показанных ранее индексов PROCESS_ PRIORITY_CLASS, устанавливающих приоритеты 4, 8, 13, 14, 6 и 10 соответственно. Затем применяется относительный приоритет потока в качестве разницы для этого базового приоритета. Например, наивысший «Highest»-поток получит базовый приоритет потока на два уровня выше, чем базовый приоритет его процесса. Далее приведено соответствие между приоритетами Windows API и ядра Windows:

Таблица 5.3. 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

Рисунок 2.2: Сответствие между приоритетами Windows API и ядра Windows

Стоит отметить, что уровень, критичный по времени, и уровень простоя (+15 и –15) называются уровнями насыщения (saturation) и представляют конкретные применяемые уровни вместо смещений. Это означает, что при положительном насыщении поток получает наивысший возможный приоритет внутри его класса приоритета, а при отрицательном насыщении — самый низкий.

Сценарии повышения приоритета (и их причины):

- Повышения связанные с завершением ожидания.

  Пытаются уменьшить время задержки между потоком, пробуждающимся по сигналу объекта. Поскольку событие, которого ждал поток, может дать информацию на данный момент времени. В противном случае информация может стать неактуальной.

- Повышение вследствие событий планировщика или диспетчера (сокращение задержек).

  В очередь потока поставлен APC-вызов, событие установлено или отправило сигнал, системное время изменилось (таймеры должны быть перезапущены), мьютекс или семафор освобождены, изменилось состояние потока.

- Повышение вследствие завершения ввода-вывода (сокращение задержек).

  Windows дает 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. Рекомендуемые значения повышения приоритета представлены в таблице.

p_pri	Текущий приоритет планирования	Используется для хранения временного приоритета для выполнения в режиме ядра
p_uspri	Приоритет режима задачи	Используется для хранения приоритета, который будет назначен процессу при возврате в режим задачи
p_cpu	Результат последнего измерения использования процессора	Содержит величину результата последнего сделанного измерения использования процессора процессом. Инициализируется нулем.
p_nice	Фактор nice, устанавливаемый пользователем	Увеличение значения приводит к уменьшению приоритета.

- Повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика).

  Потоки, владеющие окнами, получают при пробуждении дополнительное повышение приоритета на 2 уровня, из-за активности системы работы с окнами, например при поступлении сообщений. Приоритет для создания преимуществ, содействия интерактивным приложениям.

- Повышение вследствие слишком продолжительного ожидания ресурса исполняющей системы (предотвращение зависания).

  Когда поток пытается получить ресурс исполняющей системы, 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 5 секунд. Если по окончании этих 5 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, перезапуска их квантов и выполнения еще одного ожидания.

- Повышение в случае, когда готовый к запуску поток не был запущен в течение определенного времени (предотвращение зависания и смены приоритетов).

  Один раз в секунду диспетчер настройки баланса 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. 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.

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(см. рисунок 2.3).

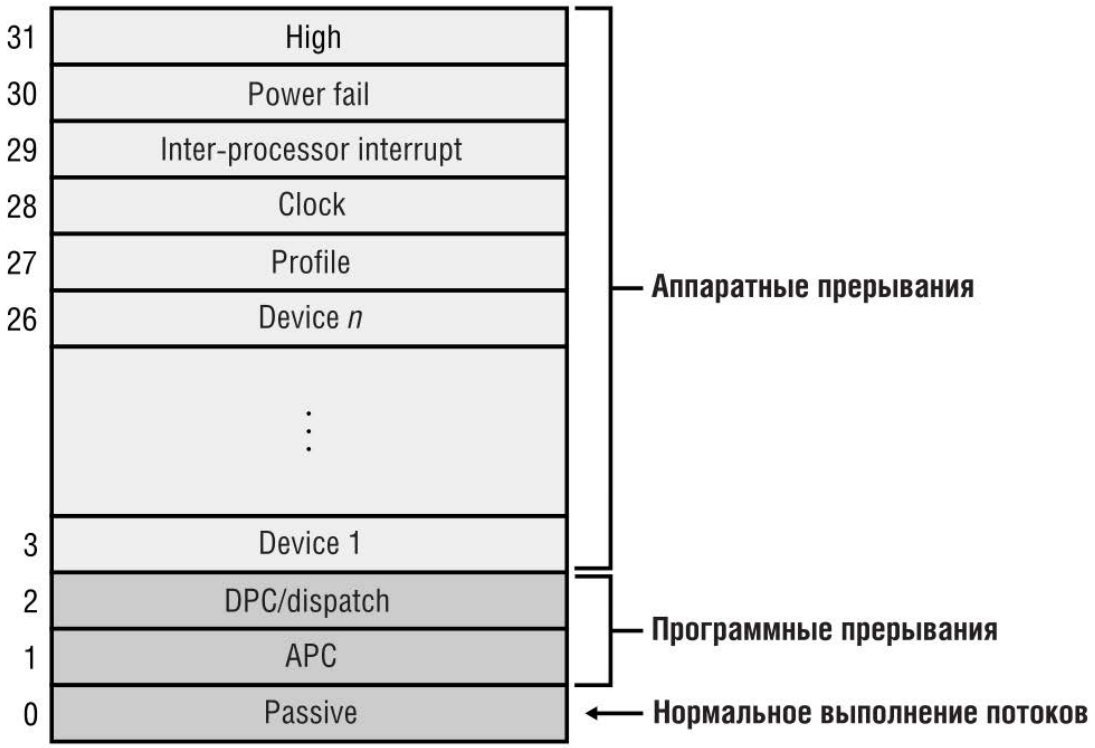

Рисунок 2.3: Уровни запросов прерываний

Прерывания обслуживаются в порядке их приоритета. Прерывания с большим 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 состо- янии 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. То- гда вышеописанный процесс повторяется ядром для обработки и этих прерываний.

Заключение

Поскольку UNIX и Windows являются системами разделения времени с динамическими приоритетами и вытеснением, обработчик прерываний от системного таймера в каждой из ОС выполняет схожие функции. Следующие задачи являются общими для каждого из обработчиков:

- декремент текущего кванта потока;
- инкремент счетчика системных тиков;
- инициализация, но не выполнение, отложенных действий, относящихся к работе планировщика, например, пересчет приоритетов.

Пересчёт динамических приоритетов в данных системах можно охарактеризовать следующим образом:

- в UNIX приоритет процесса характеризуется текущим приоритетом и приоритетом процесса в режиме задачи. Приоритет пользовательского процесса (процесса в режиме задачи) может быть динамически пересчитан в зависимости от величины использования процессора и фактора любезности; в свою очередь, приоритеты ядра являются фиксированными величинами;
- в ОС Windows при создании процесса ему назначается некоторый базовый приоритет. Приоритеты потоков вычисляется относительно базового приоритета процесса, в котором они создаются. Только приоритеты потоков пользовательского процесса могут быть динамически пересчитаны.