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»

Лабораторная работа №1 (часть 2) по дисциплине «Операционные системы»

Тема Функции системного таймера и пересчёт динамических приоритетов

Студент Сапожков А. М.

Группа ИУ7-53Б

Преподаватель Рязанова Н. Ю.

Москва — 2022 г.

1. Функции обработчика прерывания от системного таймера

1.1. Windows

По тику:

- инкремент счётчика реального времени;
- декремент кванта текущего потока на величину, равную количеству тактов процессора, произошедших за тик (если количество затраченных потоком тактов процессора достигает выделенного ему кванта, запускается обработка истечения кванта);
- декремент счётчиков времени отложенных задач;
- инициализация отложенного вызова обработчика ловушки профилирования ядра, если активен механизм профилирования ядра (объект ставится в очередь **DPC**[1]: обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания).

По главному тику:

- инициализация диспетчера настройки баланса (сбрасывается объект «событие», на котором он ожидает).

По кванту:

- инициализация диспетчеризации потоков путем постановки соответствующего объекта в очередь **DPC**.

1.2. Unix/Linux

По тику:

- инкремент счётчиков реального времени и времени, прошедшего с момента включения системы;
- декремент кванта текущего потока;
- инкремент счётчика использования процессора активным процессом — инкремент поля **c_cpu** дескриптора активного процесса до максимального значения 127;

[1]DPC — Deferred Procedure Call (отложенный вызов процедуры)

- декремент счетчиков времени до отправки на выполнение отложенного вызова (если счетчик достиг нуля, то выставляется флаг для обработчика отложенных вызовов).

По главному тику:

- инициализация отложенных вызовов функций, относящихся к работе планировщика, таких как пересчет приоритетов;

- пробуждение системных процессов, таких как **swapper** (когда необходимо загрузить в память выгруженные процессы, если есть место) и **pagedaemon** (когда необходимо выделить физическую память) (пробуждение — регистрация отложенного вызова процедуры **wakeup**, которая перемещает дескрипторы процессов из списка «спящих» в очередь готовых к выполнению);

- декремент счётчиков времени, оставшегося до посылки одного из следующих сигналов:

  SIGALRM — сигнал, посылаемый процессу по истечении времени, предварительно 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:

- посылка сигнала **SIGXCPU** активному процессу, если тот превысил выделенный ему квант процессорного времени.

2. Пересчёт динамических приоритетов

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

2.1. Windows

В ОС семейства Windows процессу при создании назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается относительный приоритет. Планирование осуществляется на основе приоритетов потоков, готовых к выполнению. Поток с более низким приоритетом вытесняется потоком с более высоким приоритетом,в тот момент когда этот поток

становится готовым к выполнению. По истечению кванта времени текущего потока, ресурс передается самому приоритетному потоку в очереди готовых к выполнению.

Каждый поток имеет динамический приоритет. Это приоритет, который планировщик использует для определения того, какой поток следует выполнить. Изначально динамический приоритет потока совпадает с базовым приоритетом процесса.

В ОС семейства Windows используется 32 уровня приоритета:

- от 0 до 15 — изменяющиеся уровни (уровень 0 зарезервирован для потока обнуления страниц);
- от 16 до 31 — уровни реального времени.

Система может повысить и понизить динамический приоритет, чтобы обеспечить скорость реагирования и отсутствие нехватки потоков на время процессора. Система не повышает приоритет потоков с базовым уровнем приоритета от 16 до 31. Только потоки с базовым приоритетом от 0 до 15 получают динамический приоритет.

Уровни приоритета потоков назначаются с двух позиций: Windows API и ядра 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).

Функция **SetPriorityClass** позволяет изменять класс приоритета процесса до одного из этих уровней.

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);
- 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.

Таким образом, 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. В таблице 1 приведено соответствие между приоритетами Windows API и ядра системы приоритета.

Таблица 1. Соответствие между приоритетами **Windows API** и ядра Windows

	real-time	**high**	**above normal**	**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, который сканирует очередь готовых процессов 1 раз в секунду. Если он обнаруживает 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.

Текущий приоритет потока в динамическом диапазоне (от 1 до 15) может быть изменён планировщиком вследствие следующих причин:

- повышение приоритета после завершения операций ввода-вывода;
- повышение приоритета владельца блокировки;

- повышение приоритета вследствие ввода из пользовательского интерфейса;
- повышение приоритета вследствие длительного ожидания ресурса исполняющей системы;
- повышение приоритета вследствие ожидания объекта ядра;
- повышение приоритета в случае, когда готовый к выполнению поток не был запущен в течение длительного времени;
- повышение приоритета проигрывания мультимедиа службой планировщика **MMCSS**.

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового путем вычитания всех его повышений. В таблице 2 приведены рекомендуемые значения повышения приоритета для устройств ввода-вывода.

Таблица 2. Рекомендуемые значения повышения приоритета.

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

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;
- игры;
- распределение;
- захват;

- воспроизведение;
- задачи администратора многоэкранного режима.

Одно из наиболее важных свойств для планирования потоков — категория планирования — первичный фактор определяющий приоритет потоков, зарегистрированных в **MMCSS**. Различные категории планирования представленны в таблице 3.

Таблица 3. 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

Функции драйвера MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, который соответствует категории планирования. Потом их приоритет снижается до уровня, соответствующего категории планирования Exhausted, для того, чтобы другие потоки тоже могли получить ресурс.

2.1.1. Уровни запросов программных прерываний (IRQL)

Хотя контроллеры прерываний устанавливают приоритетность прерываний, Windows устанавливает свою собственную схему приоритетности прерываний, известную как уровни запросов прерываний (IRQL). В ядре IRQL-уровни представлены в виде номеров от 0 до 31 на системах x86 и в виде номеров

от 0 до 15 на системах x64 и IA64, где более высоким номерам соответствуют прерывания с более высоким приоритетом. Хотя ядро определяет для программных прерываний стандартный набор IRQL-уровней, HAL отображает номера аппаратных прерываний на IRQL-уровни. На рисунке 1 показаны IRQL-уровни для архитектуры x86, а на рисунке 2 показаны IRQL-уровни для архитектур x64 и IA64.

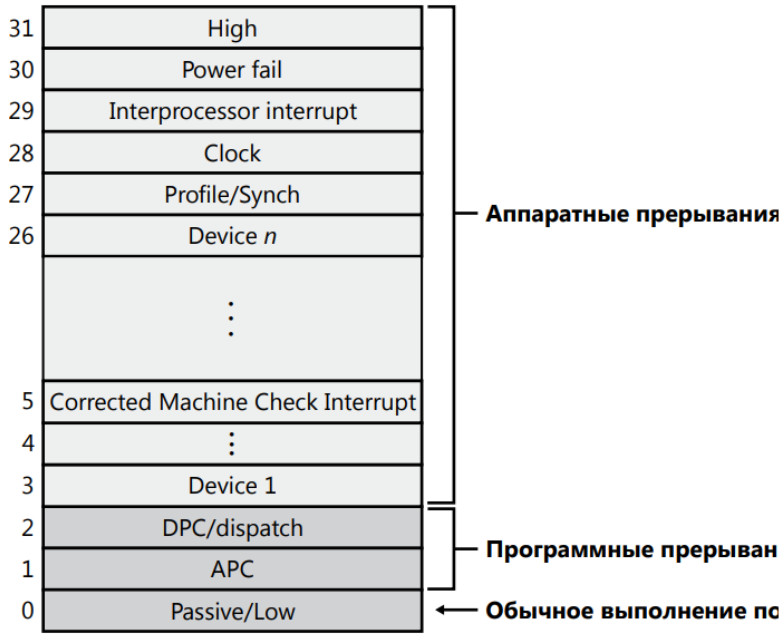

Рис. 1. Уровни запросов прерываний (IRQL) для архитектуры x86

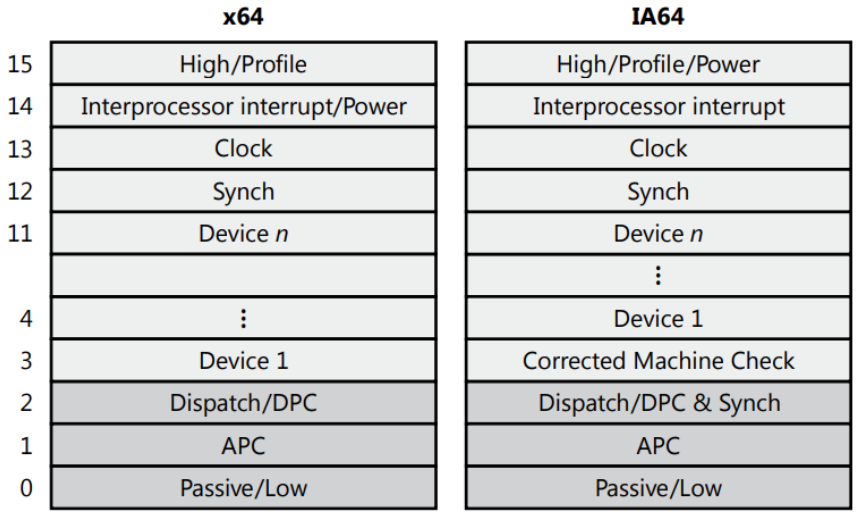

Рис. 2. Уровни запросов прерываний (IRQL) для архитектур x64 и IA64

Прерывания обслуживаются в порядке их приоритета, и прерывания с более высоким уровнем приоритета получают преимущество в обслуживании.

При возникновении прерывания с высоким уровнем процессор сохраняет состояние прерванного потока и запускает связанный с прерыванием диспетчер системных прерываний. Тот, в свою очередь, поднимает IRQL и вызывает процедуру обработки прерывания. После выполнения этой процедуры диспетчер прерываний понижает IRQL-уровень процессора до значения, на котором он был до возникновения прерывания, а затем загружает сохраненное состояние машины. Прерванный поток продолжает выполнение с того места, в котором оно было прервано. Когда ядро понижает IRQL, могут реализоваться те прерывания с более низким уровнем приоритета, которые были замаскированы. Если так и происходит, ядро повторяет процесс для обработки новых прерываний.

Уровни приоритетов IRQL имеют совершенно другое значение, чем приоритеты, используемые при планировании потоков. Приоритет планирования является атрибутом потока, а IRQL является атрибутом источника прерывания, такого как клавиатура или мышь. Кроме того, у каждого процессора есть установка IRQL, которая меняется при выполнении кода операционной системы.

Установка IRQL каждого процессора определяет, какие прерывания данный процессор может получать. IRQL-уровни также используются для синхронизации доступа к структуре данных режима ядра. Как только запускается поток режима ядра, он повышает или понижает IRQL процессора либо напрямую, путем вызова функций KeRaiseIrql и KeLowerIrql, либо, что случается чаще, опосредованно, через вызовы функций, которые запрашивают объекты ядра, используемые для синхронизации. Как показано на рисунке 3, прерывания, поступающие от источника с IRQL, превышающим текущий уровень, прерывают работу процессора, а прерывания, поступающие от источников с IRQL-уровнями равными или ниже текущего уровня, маскируются до тех пор, пока выполняющийся поток не понизит IRQL.

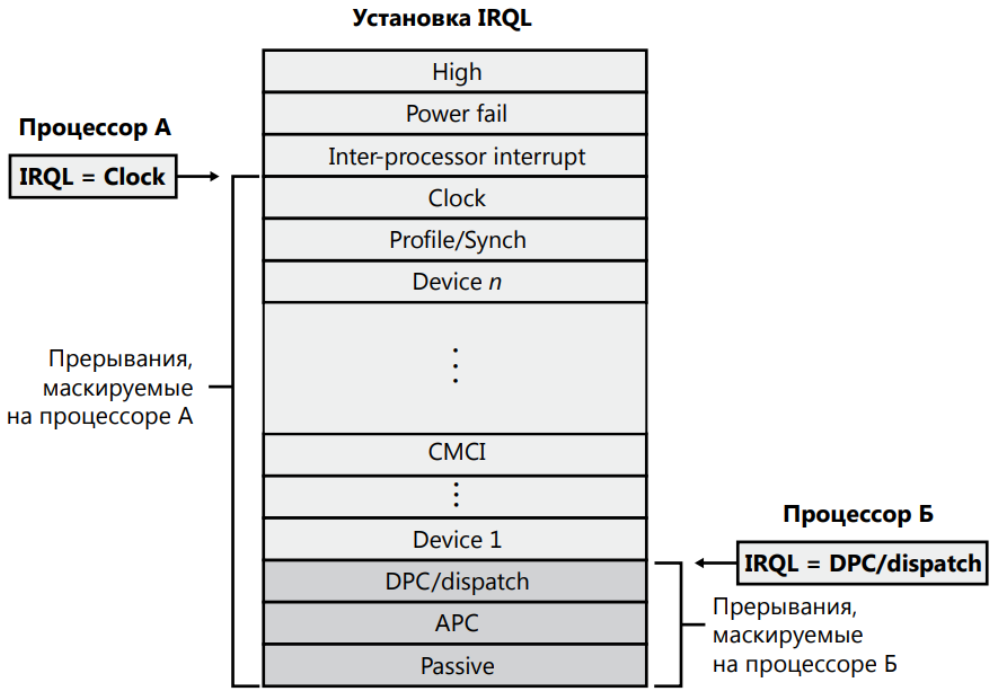

Рис. 3. Маскирование прерываний

2.2. Unix/Linux

В современном Unix ядро является вытесняющим — процесс в режиме ядра может быть вытеснен более приоритетным процессом, так же находящимся в режиме ядра. Это было сделано для того, чтобы система могла обслуживать процессы реального времени, такие как воспроизведение видео или аудио.

Согласно приоритетам процессов и принципу вытесняющего циклического планирования формируется очередь готовых к выполнению потоков. В первую очередь выполняются процессы с большим приоритетом. Процессы с одинаковыми приоритетами выполняются в течении кванта времени — циклически, друг за другом. В случае, если поток, имеющий более высокий приоритет поступает в очередь готовых к выполнению процессов, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному процессу.

Приоритет — это целое число, находящееся в диапазоне от 0 до 127. Чем меньше значение, тем выше приоритет процесса. Приоритеты ядра варьируются от 0 до 49, а приоритеты прикладных задач от 50 до 127. Приоритеты ядра являются статическими величинами, а приоритеты прикладных задач могут изменяться во времени в зависимости от двух факторов: фактора «любезности» и фактора утилизации.

Фактор «любезности» — это целое число в диапазоне от 0 до 39 (по умол-

чанию 20). Чем меньше значение фактора «любезности» процесса, тем выше приоритет процесса. Фактор «любезности» процесса может быть изменен с помощью системного вызова **nice**, но только суперпользователем. Фоновым процессам задаются более высокие значения фактора «любезности».

Фактор утилизации определяется последней измеренной величиной использования процессора. Этот фактор позволяет системе динамически изменять приоритет процесса.

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

- **p_pri** — текущий приоритет планирования;
- **p_usrpri** — приоритет процесса в режиме задачи;
- **p_cpu** — результат последнего измерения степени загруженности процессора (процессом);
- **p_nice** — фактор любезности, устанавливаемый пользователем.

Когда процесс 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, а **p_usrpri** будет использоваться для хранения приоритета который будет назначение когда процесс вернется в режим задачи.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. В тот момент когда процесс просыпается, после того как был блокирован в системном вызове, ядро устанавливается приоритет сна в поле **p_pri** — это значение приоритета в диапазоне от 0 до 49, зависящее от события или ресурса по которому произошла блокировка. В таблице 4 приведены значения приоритетов сна для систем **4.3BSD**.

Таблица 4: Приоритеты сна в ОС 4.3 BSD

Событие	**Приоритет 4.3BSD UNIX**	**Приоритет SCO UNIX**
Ожидание загрузки в память сегмента/страницы	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 после **p_cpu** инициализируется нулём. На каждом тике обработчик таймера увеличивает это поле для текущего процесса на единицу, до максимального значения, которое равно 127. Каждую секунду обработчик прерывания инициализирует отложенный вызов процедуры **schedcpy()**, которая уменьшает значение **p_cpu** каждого процесса исходя из фактора «полураспада». В системе **4.3BSD** фактор полураспада рассчитывается по формуле (1):

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

где **load_average** — среднее количество процессов, находящихся в состоянии готовности к выполнению (за последнюю секунду).

Приоритеты для режима задачи всех процессов в процедуре **schedcpy()** пересчитываются по формуле (2):

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2 \cdot p\_nice \tag{2}$$

где **PUSER** — базовый приоритет в режиме задачи, равный 50.

Если процесс в последний раз использовал большое количество процессорного времени, его **p_cpu** будет увеличен. Это приведёт к росту значения **p_usrpri** и понижению приоритета. Чем дольше процесс находится в очереди на исполнение, тем больше фактор полураспада уменьшает его **p_cpu**, что приводит к повышению его приоритета. Данная схема предотвращает бесконечное откладывание низкоприоритетных процессов. Применение такой схемы более предпочтительно для процессов, которые осуществляют много операций ввода-вывода, и менее предпочтительно для процессов, производящих много вычислений.

Заключение

Обработчики прерывания от системного таймера в защищенном режиме в системах Unix и Windows выполняют одинаковые действия:

- выполняют декремент счетчиков времени: таймеров, счетчиков времени отложенных действий, будильников реального времени;

- выполняют декремент кванта текущего процесса в Linux и декремент текущего потока в Windows;
- инициализируют отложенные действия, относящиеся к работе планировщика, такие как пересчёт приоритетов.

Обе системы являются системами разделения времени с динамическими приоритетами и вытеснением процессов. Такой подход позволяет поддерживать процессы реального времени, такие как воспроизведение аудио и видео. Пересчёт динамических приоритетов пользовательских процессов выполняется для того, чтобы не допустить их бесконечного откладывания.