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 № 4, часть 2
по дисциплине «Операционные системы»

Студент Бугаков И. С.

Группа ИУ7-54Б

Преподаватель Рязанова Н. Ю.

Москва, 2025

1 Функции обработчика прерывания от системного таймера

1.1 UNIX/Linux

По тику:

— инкремент системных таймеров (например, переменной lbolt, хранящей количество тиков, осчитанных с момента загрузки системы);

— декрмент счетчиков времени до отправления на выполнение отложенного вызова (флаг для обработчика отложенных вызовов выставляется, если счетчик достиг нуля);

— декремент кванта текущего потока.

По главному тику:

— инициализация (установка флага обработчика отложенных вызовов, при наличии ожидающих вызовов) отложенных вызовов функций, связанных с работой планировщика, например, пересчет приоритетов;

— регистрация отложенного вызова процедуры wakeup для системных процессов, например swapper и kswapd;

— декремент счетчика времени до посылки одного из сигналов:

- SIGALARM — посылается процессу по истечении времени, заданного предварительно функцией alarm;
- SIGPROF — посылается процессу по истечении времени, заданного в таймере профилирования (общее время выполнения процесса);
- SIGVTALRM — посылается процессу по истичении времени, заданного в «виртуальном» таймере (время выполнения процесса в режиме задачи).

По кванту:

— отправка сигнала SIGXCPU процессу по истечении выделенного ему кванта (при получении данного сигнала процесс прерывает свое выполнение, либо диспетчер выделяет ему еще один квант).

1.2 Windows

По тику:

— инкремент счетчика системного времени;

— декремент счетчиков времени отложенных задач;

— декремент кванта текущего потока.

По главному тику:

— раз в секунду инициализация диспетчера настройки баланса, путем сброса объекта «событие», на котором он ожидает.

По кванту:

— инициализация диспетчеризации потоков — постановка соответствующего объекта в очередь DPC.

2 Пересчет динамических приоритетов

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

2.1 UNIX/Linux

В традиционном UNIX ядро является строго не вытесняющим — процесс в режие ядра не может быть вытеснен другим более приоритетным процессом. В современных система UNIX ядро является вытесняющим, т. е. процесс в режиме ядра может быть вытеснен более приоритетным процессом. Вытесняющее ядро позволяет системе обслуживать прцоессы реального времени.

2.1.1 Приоритеты процессов

Очередь готовых к выполнению процессов формируется в соответствии с приоритетами и приниципом вытесняющего циклического планирования: процессы с одинаковыми приоритетами выполняются в течении кванта времени друг за другом циклически. Если в очередь готовых к выполнению процессов поступает более приоритетный, чем выполняющийся сейчас, процесс, планировщик вытесняет текующий процесс и предоставляет квант более приоритетному процессу.

Приоритет в UNIX задается целым числом в диапазоне от 0 до 127. Чем меньше значение, тем выше приоритет процесса. Приоритеты 0 – 49 зарезервированы для процессов ядра, приоритеты 50 – 127 предназначаются прикладным процессам. Т. к. пересчитываться могут только приоритеты пользовательских процессов, приоритеты процессов ядра являются статическими. Приоритеты прикладных процессов могут изменяться в зависимости от фактора «любезности» и степени загруженности процессора в момент последнего обновления.

Фактор «любезности» процесса — целое число в диапазоне от 0 до 39. Чем меньше значение этого фактора, тем выше приоритет процесса. По умолчанию значение фактора «любезности» — 20. Фактор «любезности» процесса может быть изменен суперпользователем системным вызовом **nice**. Фоновые процессы обладают большими значениями фактора «любезности».

В дескрипторе процесса **proc** содержатся поля, которые относятся к приоритету процесса:

— **p_pri** — текущий приоритет планирования;

— **p_usrpri** — приоритет процесса в режиме задачи;

— **p_cpu** — результат последнего измереня степени загруженности процессора;

— **p_nice** — фактор любезности, устанавливаемый пользователем.

Для процесса находящегося в режиме задачи, значения **p_pri** и **p_usrpri** равны. Значение

текущего приоритета **p_pri** может быть временно повышено планировщиком для выполнения процесса в режиме ядра, при этом **p_usrpri** будет использован для хранения приоритета процесса при возвращении в режим задачи.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может быть блокирован. Когда процесс просыпается, после блокировки в системном вызове, ядро устанавливает приоритет сна в поле **p_pri** — значение от 0 до 49, в зависимости от того, по какому событию или ресурсу произошла блокирвка. Такой процесс будет приоритетнее процессов в режиме задачи. В таблице 2.1 приведены приоритеты сна для систем 4.3 BSD и SCO UNIX.

Таблица 2.1 — Приоритеты сна в ОС 4.3 BSD и SCO UNIX

Событие	**Приоритет 4.3 BSD**	**Приоритет SCO UNIX**
Ожидание загрузки в память сегомента/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 полю **p_cpu** присваивается значение 0. На каждом тике обработчик прерывания таймера икрементирует это поле для текузего процесса, до максимального значения, равного 127.

Каждую секунду, обработчик прерывания таймера инициализирует отложенный вызов процедуры **schedcpy**, которая уменьшает значение **p_cpu** в соответстви с фактором полураспада. В 4.3 BSD для расчета этого фактора применяется формула:

$$decay = \frac{2 \cdot load_average}{2 \cdot load_average + 1}$$

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

Приоритеты всех процессов в режиме задачи в процедуре **schedcpy** пересчитываются по формуле:

$$p_usrpri = PUSER + \frac{p_cpu}{4} + 2 \cdot_n ice$$

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

Если процесс в последний раз использовал большое количество процессорного времени, поле **p_cpu** его дексриптора будет увеличено. Это приведёт к росту значения **p_usrpri** и понижению приоритета. Чем дольше процесс находится в очереди на исполнение, тем больше фактор полураспада уменьшает значение **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.

2.1.2 Потоки в UNIX

В современном UNIX потоки являются единицей диспетчеризации. Для приоритизации потоков используется три класса планирования.

Классы планирования:

— Deadline: Имеет наивысший приоритет. Используется для задач с жёсткими временными ограничениями;

— Real-Time: Второй по приоритету класс, предназначен для задач реального времени;

— Fair: Предназначен для прикладных задач.

Приоритеты:

1) NICE: используется для потоков класса Fair и варьируется от -20 (максимальный приоритет) до +19 (минимальный приоритет). Чем выше значение, тем больше поток делится процессорным временем;

2) Real-Time приоритеты: для потоков класса Real-Time приоритеты варьируются от 1 до 99. NICE здесь не используется;

3) Priority (sched_priority): определяет политику и приоритет для потоков класса Real-Time и Deadline. Для потоков класса Deadline приоритет всегда равен 0.

Алгоритмы планирования:

— Completely Fair Scheduler – CFS обеспечивает равномерное распределение процессорного времени, моделируя "идеальный многозадачный CPU". Процессы сортируются по виртуальному времени (vruntime) в красно-чёрном дереве. Задача с минимальным vruntime получает процессорное время.

— Earliest Eligible Virtual Deadline First — EEVDF заменил CFS в 2023 году. Алгоритм ориентирован на своевременное выполнение задач с дедлайнами. Задачи сортируются по дедлайнам, а параметр eligibility определяет, имеет ли задача право на выполнение.

2.2 Windows

В ОС семейства Windows процессу при создании присваивается базовый приоритет. Относительно базового приоритета высчитывается относительный приоритет. Планирование осуществляется на основе приоритетов потоков, готовых к выполнению. Потоки с более низким приоритетом могут быть вытеснены потоками с более высокими приоритетами, когда те будут готовы к выполнению. По истечению кванта времени текущего потока, ресурс передается самому приоритетному потоку в очереди готовых к выполнению.

В ОС семейства Windows существует 32 уровня приоритета:

— от 0 до 15 — изменяющиеся уровни (уровень 0 зарезервирован для потока обнуления страниц);

— от 16 до 31 — уровни реального времени.

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 2.2 приведено соответствие между приоритетами Windows API и ядра системы приоритета.

Таблица 2.2 — 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 (семафора, события);

— 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 2.3 приведены рекомендуемые значения повышения приоритета для устройств ввода-вывода.

Таблица 2.3 — Рекомендуемые значение 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.

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

2.2.1 Уровень запроса прерывания (IRQL)

Windows устанавливает свою собственную схему приоритетности прерываний, известную как 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. На рисунках 2.1, 2.2 показаны IRQL для архитектур x86, x64 и IA64 соответственно.

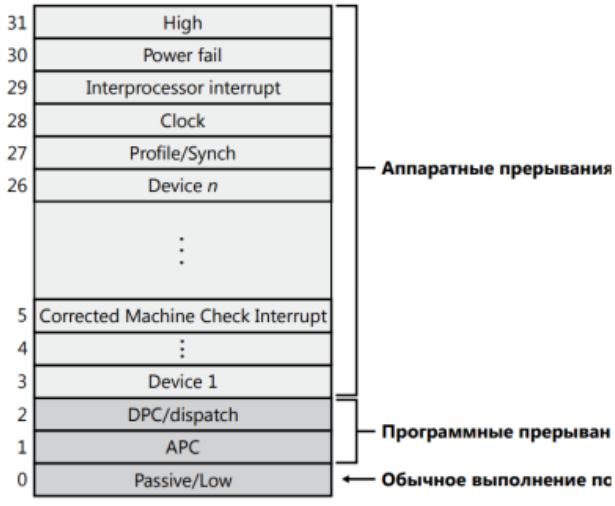

Рисунок 2.1 — Уровни запросов прерываний для архитектуры x86

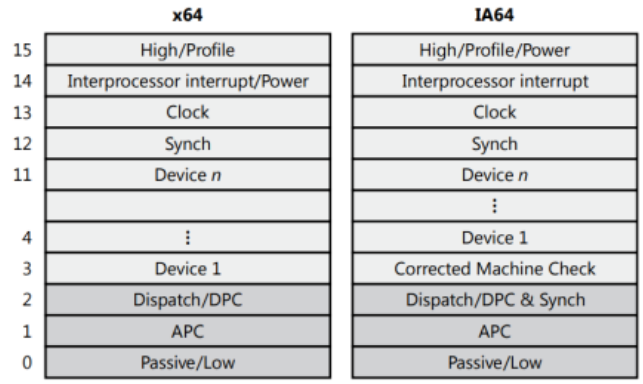

Рисунок 2.2 — Уровни запросов прерываний IRQL для архитектур x64 и IA64

3 Выводы

Обработчики прерывания от системного таймера в защищенном режиме в системах Unix и Windows следующие 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. Динамические приоритеты пользовательских процессов позволяют исключить проблему бесконечного откладывания. Вытеснение процессов дает возможность поддерживать процессы реального времени, такие как аудио и видео.