



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, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 1

по курсу «Операционные системы»

на тему: «Обработчик прерывания от системного таймера»

Студент ИУ7-55Б
(Группа)

(Подпись, дата)

Романов С. К.
(И. О. Фамилия)

Преподаватель

(Подпись, дата)

Рязанова Н. Ю.
(И. О. Фамилия)

2022 г.

Оглавление

1	Функции обработчика прерывания от системного таймера в защищённом режиме	3
1.1	Функции обработчика прерывания от системного таймера в защищённом режиме для ОС семейства UNIX/Linux	3
1.2	Функции обработчика прерывания от системного таймера в защищённом режиме для ОС семейства Windows	4
2	Пересчёт динамических приоритетов	6
2.1	Пересчёт динамических приоритетов в операционных системах UNIX/Linux	6
2.2	Пересчет динамических приоритетов в операционных системах семейства Windows	9
2.2.1	MMCSS	12
	Вывод	14

1 Функции обработчика прерывания от системного таймера в защищённом режиме

1.1 Функции обработчика прерывания от системного таймера в защищённом режиме для ОС семейства UNIX/Linux

Обработчик прерывания от системного таймера **по тик** выполняет следующие задачи:

- инкрементирует счетчик тиков аппаратного таймера;
- декрементирует квант текущего потока;
- обновляет статистику использования процессора текущим процессом;
- обновляет часы и другие таймеры системы;
- декрементирует счетчик времени до отправления на выполнение отложенного вызова (если счетчик достиг нуля, то выставление флага для обработчика отложенного вызова).

Обработчик прерывания от системного таймера **по главному тик** выполняет следующие задачи:

- регистрирует отложенные вызовы функций, относящиеся к работе планировщика, такие как пересчет приоритетов;
- пробуждает в нужные моменты системные процессы, такие как `swapper` и `pagedaemon`. Под пробуждением понимается регистрация отложенного вызова процедуры `wakeup`, которая перемещает дескрипторы процессов из списка “спящих” в очередь готовых к выполнению.
- декрементирует счётчик времени, оставшегося времени до отправки 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.

Обработчик прерывания от системного таймера **по кванту** выполняет следующие задачи:

- посылает текущему процессу сигнал **SIGXCPU**, если тот превысил выделенную ему квоту использования процессора.

1.2 Функции обработчика прерывания от системного таймера в защищённом режиме для ОС семейства Windows

Обработчик прерывания от системного таймера **по тик** выполняет следующие задачи:

- инкрементирует счётчика системного времени;
- декрементирует кванта текущего потока на величину, равную количеству тактов процессора, произошедших за тик (если количество затраченных потоком тактов процессора достигает квантовой цели, запускается обработка истечения кванта);
- декрементирует счётчиков времени отложенных задач;
- если активен механизм профилирования ядра, инициализирует отложенный вызов обработчика ловушки профилирования ядра путем постановки объекта в очередь **DPC**: обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

Обработчик прерывания от системного таймера **по главному тик** выполняет следующие задачи:

- освобождает объект “событие”, который ожидает диспетчер настройки баланса.

Обработчик прерывания от системного таймера **по кванту** выполняет следующие задачи:

- инициализирует диспетчеризацию потоков путем постановки соответствующего объекта в очередь DPC.

2 Пересчёт динамических приоритетов

Как в ОС семейства UNIX/Linux так и в ОС семейства Windows могут динамически пересчитываться только приоритеты пользовательских процессов.

2.1 Пересчёт динамических приоритетов в операционных системах UNIX/Linux

В современных системах UNIX/Linux ядро является вытесняющим – процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Ядро было сделано вытесняющим для того, чтобы система могла обслуживать процессы реального времени, такие как аудио и видео.

Очередь готовых к выполнению процессов формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования: в первую очередь выполняются процессы с большим приоритетом, а процессы с одинаковыми приоритетами выполняются в течении кванта времени циклически друг за другом. Если процесс, имеющий более высокий приоритет, поступает в очередь готовых к выполнению, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному.

Приоритет представляет собой целое число из диапазона от 0 до 127. Чем меньше число, тем выше приоритет:

- в диапазоне от 0 до 49 находятся приоритеты ядра;
- в диапазоне от 50 до 127 – приоритеты прикладных задач.

Приоритеты ядра являются фиксированными величинами.

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

- фактор любезности;
- последней измеренной величины использования процессора.

Фактор любезности – целое число в диапазоне от 0 до 39 со значением 20 по умолчанию. Чем меньше значение фактора любезности, тем выше

приоритет процесса. Фоновым процессам автоматически задаются более высокие значения этого фактора. Фактор любезности процесса может быть изменен суперпользователем с помощью системного вызова `nice`.

Дескриптор процесса `proc` содержит следующие поля, относящиеся к приоритету:

- `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.

У процесса, находящегося 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 (при этом `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. Событие и связанное с ним значение приоритета сна в системе 4.3BSD описывает таблица 2.1.

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).

$$decay = \frac{2 \cdot load_average}{2 \cdot load_average + 1} \quad (2.1)$$

Таблица 2.1 – Таблица приоритетов в системе 4.3BSD

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

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).

$$p_usrpri = PUSER + \frac{p_cpu}{2} + 2 \cdot p_nice \quad (2.2)$$

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 приведёт к росту значения `p_usrpri` и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на исполнение, тем больше фактор полураспада уменьшает его `p_cpu`, что приводит к повышению его приоритета. Такая схема предотвращает зависание низкоприоритетных процессов по вине операционной системы. Её применение предпочтительнее процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

2.2 Пересчет динамических приоритетов в операционных системах семейства Windows

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

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

Раз в секунду диспетчер настройки баланса сканирует очередь готовых потоков. Если обнаружены потоки, ожидающие выполнения более 4 секунд, диспетчер настройки баланса повышает их приоритет до 15. Как только квант истекает, приоритет потока снижается до базового приоритета. Если поток не был завершён за квант времени или был вытеснен потоком с более высоким приоритетом, то после снижения приоритета поток возвращается в очередь готовых потоков.

Чтобы минимизировать расход процессорного времени, диспетчер настройки баланса сканирует лишь 16 готовых потоков. Кроме того, диспетчер повышает приоритет не более чем у 10 потоков за один проход: обнаружив 10 потоков, приоритет которых следует повысить, он прекращает сканирование. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз. Наличие 10 потоков, приоритет которых следует повысить, говорит о необычно высокой загруженности системы.

В Windows используется 32 уровня приоритета: целое число от 0 до 31, где 31 — наивысший приоритет, из них:

- от 16 до 31 — уровни реального времени;
- от 0 до 15 — динамические уровни, уровень 0 зарезервирован для потока обнуления страниц.

Уровни приоритета потоков назначаются Windows API и ядром 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).

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.

Соответствие между приоритетами Windows API и ядра системы приведено в таблице 2.2.

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

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

- повышение вследствие событие планировщика или диспетчера;
- повышение приоритета владельца блокировки;
- повышение приоритета после завершения ввода/вывода (см. таблицу 2.3);
- 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 мультимедиа службой планировщика MMCSS.

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового приоритета путем вычитания всех повышений.

Таблица 2.3 – 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

2.2.1 MMCSS

Мультимедийные потоки должны выполняться с минимальными задержками. Эта задача решена в Windows путем повышения приоритетов мультимедийных потоков драйвером **MultiMedia Class Scheduler Service** (MMCSS). Повышение приоритетов мультимедийных потоков происходит следующим образом: приложения, которые реализуют воспроизведение мультимедийного контента, указывают драйверу MMCSS задачу из следующего списка:

- аудио;
- захват;
- распределение;
- игры;
- воспроизведение;
- задачи администратора многоэкранного режима.

Одно из наиболее важных свойств для планирования потоков называется категорией планирования — является первичным фактором, определяющим приоритет потоков, зарегистрированных с MMCSS. В таблице 2.4 показаны различные категории планирования.

Функции MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, соответствующего их категориям планирования.

Таблица 2.4 – 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

Затем, их приоритет снижается до уровня, соответствующего категории **Exhausted**, чтобы другие потоки также могли получить ресурс.

Вывод

Функции обработчика прерывания от системного таймера в защищенном режиме для семейства ОС **Windows** и для семейства ОС **UNIX/Linux** очень похожи по своим действиям. Они выполняют схожие задачи:

- инициализируют (но не выполняют) отложенные действия, относящиеся к работе планировщика, такие как пересчет приоритетов;
- выполняют декремент счетчиков времени: часов, таймеров, будильников реального времени, счетчиков времени отложенных действий.
- выполняют декремент кванта (текущего процесса в **Linux**, текущего потока в **Windows**).

Обе системы являются системами разделения времени с динамическими приоритетами и вытеснением, пересчёт динамических приоритетов в данных системах можно описать следующим образом:

- В **UNIX/Linux** 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 фактора любезности и величины использования процессора, в то время как приоритеты ядра являются фиксированными величинами.
- При создании процесса в **Windows**, ему назначается приоритет, обычно называемый базовым. Приоритеты потоков определяются относительно приоритета процесса, в котором они создаются. Приоритет потока пользовательского процесса может быть пересчитан динамически.