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»

на тему: «Прерывание таймера в Windows и Unix»

Студент ИУ7-53Б _____ Г. А. Миронов
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Н. Ю. Рязанова
(Подпись, дата) (И. О. Фамилия)

2021 г.

СОДЕРЖАНИЕ

1 Функции обработчика прерываний от системного таймера

1.1 Некоторые теоретические сведения

Тик – период времени между двумя последующими прерываниями таймера.

Основной тик – период времени, равный `N` тикам таймера (число `N` зависит от конкретной системы).

Квант – период времени, отведенный планировщиком процессу для выполнения.

1.2 UNIX-системы

1.2.1 По тику

- инкремент счетчика тиков аппаратного таймера по необходимости;
- инкремент часов и других таймеров системы;
- декремент счетчика времени, значение которого показывает оставшееся время до запуска обработчика отложенных вызовов, и выставление флага для обработчика отложенных вызовов при достижении счетчиком нулевого значения;
- обновление статистики использования процессора текущим процессом;
  - инкремент поля `p_cpu` дескриптора текущего процесса (до максимального значения 127);
- декремент кванта текущего процесса.

1.2.2 По главному тику

- пробуждение в нужные моменты системных процессов, таких как `swapper` и `pagedaemon`. Пробуждение означает регистрацию отложенного вызова процедуры `wakeup`, которая перемещает дескрипторы процессов из списка "спящих" в очередь готовых к выполнению;

- декремент счётчика времени, который отвечает за время, оставшееся до посылки одного из сигналов (так называемых *будильников*):
  - `SIGALRM` – сигнал будильника реального времени, отправляемый по истечении заданного промежутка действительного времени;
  - `SIGPROF` – сигнал будильника профиля процессора, измеряющего время работы процессора;
  - `SIGVTALRM` – сиггад будильника виртуального времени, измеряющего только время работы процесса в режиме задачи.

1.2.3 По кванту

- при превышении текущим процессом выделенного ему кванта, отправка сигнал `SIGXCPU` – превышение лимита процессорного времени этому процессу.

1.3 Windows-системы

1.3.1 По тику

- инкремент счетчик системного времени;
- декремент счетчика времени отложенных задач;
- декремент остатка кванта текущего процесса;
  - декрементирование происходит на величину, равную количеству тактов процессора, произошедших за тик;
  - при достижении количеством затраченных тактов процессора квантовой цели, запускается обработка истечения кванта;
- в случае, если активен механизм профилирования ядра, инициализация отложенного вызова обработчика ловушки профилирования ядра. путём постановки объекта в очередь **DPC**.

1.3.2 По главному тику

- инициализация диспетчера настройки баланса путем сбрасывания объекта "событие", на котором он ожидает;

– диспетчер настройки баланса – это системный поток, создаваемый ядром, используемый для инициализации различных событий, связанных с планированием и управлением памятью.

1.3.3 По кванту

- инициализация диспетчеризации потоков – постановка соответствующего объекта в очередь **DPC**.

2 Пересчет динамических приоритетов

2.1 Некоторые теоретические сведения

Динамические приоритеты – это приоритеты пользовательских процессов.

2.2 UNIX-системы

В современных системах `UNIX` ядро является вытесняющим – процесс в режиме ядра может быть вытеснен более приоритетным процессом, находящимся в режиме ядра. Данная функциональность необходима, чтобы система могла обслуживать процессы реального времени, к примеру: видео и аудио.

Очередь готовых к выполнению процессов формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования. В первую очередь выполняются процессы, имеющие больший приоритет. Процессы с одинаковыми приоритетами выполняются циклически друг за другом, в течение кванта времени. В случае, если процесс с более высоким приоритетом поступает в очередь готовых к выполнению процессов, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному процессу.

2.2.1 Приоритеты процессов

Приоритет процесса в UNIX задается числом в диапазоне от 0 до 127, причем, чем меньше число, тем выше приоритет процесса.

Приоритеты от 0 до 49 заврезервированы ядром операционной системы, в связи с чем прикладные процессы могут обладать приоритетом от 0 до 127.

Структура `proc` содержит следующие поля, относящиеся к приоритетам:

- `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.

Планировщик использует `p_pri` для принятия решения о том, какой процесс направить на выполнение. У процесса, находящегося в режиме задачи, значения `p_pri` и `p_usrpri` равны. Текущее значение приоритета `p_pri` может быть повышено планировщиком для выполнения процесса в режиме ядра, а `p_usrpri` будет использоваться для хранения приоритета, который будет назначен при возврате в режим задачи. В отличие от приоритетов ядра, являющихся фиксированными величинами, приоритеты прикладных задач могут изменяться во времени в зависимости от следующих факторов:

- *фактор любезности* – это целое число в диапазоне от 0 до 39 (по умолчанию 20). Приоритет процесса тем выше, чем меньше значение фактора любезности процесса. Фактор любезности процесса может быть изменен с помощью системного вызова `nice`, однако только суперпользователь (`root`) может определять увеличение приоритета. Фоновым процессам задаются более высокие значения фактора любезности;

- *фактор утилизации* – фактор, который определяется степенью последней загруженности `CPU` процессом (последняя измеренная величина использования процессора).

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 устанавливает в поле `p_pri` приоритет сна – это значение приоритета в диапазоне от 0 до 49, зависящее от события или ресурса, по которому произошла блокировка. В таблице 2.1 приведены значения приоритетов сна для систем **4.3BSD**.

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
TTPOPRI	29	Ожидание вывода с терминала
PWAIT	30	Ожидание завершения процесса потомка
PLOCK	35	Консультативное ожидание блокированного ресурса
PSLEP	40	Ожидание сигнала

При создании процесса поле `p_cpu` инициализируется нулём. На каждом тике обработчик таймера увеличивает это поле для текущего процесса на единицу, до максимального значения, равного 127. Более того, 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 (2.1):

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

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 (2.2):

$$p\_usrpri = PUSER + \frac{p\_cpy}{4} + 2 * p\_nice \tag{2.2}$$

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, 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 бесконечное откладывание низ-

кориоритетных процессов по вине операционной системы. Ее применения предпочтительно процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

2.3 Windows-системы

В системах Windows реализовано вытесняющее планирование на основе уровней приоритета, при которой выполняется готовый поток с наивысшим приоритетом.

Если поток с более высоким приоритетом готов к выполнению, текущий поток вытесняется планировщиком, даже если квант текущего не истек.

Диспетчер настройки баланса сканирует очередь готовых потоков один раз в секунду в поиске тех из них, которые находятся в состоянии ожидания (т.е. не были запущены) около 4 секунд. Если такой поток будет найден, диспетчер настройки баланса повышает его приоритет до 15 единиц и устанавливает квантовую цель эквивалентной тактовой частоте процессора при подсчете 3 квантовых 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.

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 за планирование отвечает совокупность процедур ядра, называемая диспетчером ядра. диспетчеризация может быть вызвана, если:

- поток готов к выполнению;
- истек квант текущего потока;
- поток завершается или переходит в состояние ожидания;

- изменился приоритет потока;
- изменилась привязка потока к процессу.

2.3.1 Приоритеты процессов

В системе предусмотрено 32 уровня приоритетов:

- уровни реального времени – от 16 до 31;
- динамические уровни – от 1 до 15;
- системный уровень – 0 (зарезервирован для потока обнуления страниц).

Уровни приоритета потоков назначаются исходя из двух разных позиций: `Windows API` и ядра операционной системы. `Windows API` систематизирует процессы по классу приоритета, который им присваивается при создании:

- реального времени – `real-time` (4);
- высокий – `high (3)`;
- выше обычного – `above normal (7)`;
- обычный – `normal (2)`;
- ниже обычного – `below normal (5)`;
- 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)`;
- простоя – `idle (-15)`.

Относительный приоритет – это приращение к базовому приоритету процесса. Процесс обладает только базовым приоритетом, тогда как поток имеет базовый, который наследуется от приоритета процесса, и текущий 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 ОС

Класс приоритета/ Отн. приоритет	**realtime**	**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

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

- повышение приоритета после завершения операций ввода-вывода;
- повышение приоритета вследствие ввода из пользовательского интерфейса;
- повышение приоритета владельца блокировки;
- повышение приоритета вследствие ожидания объекта ядра;
- повышение приоритета вследствие длительного ожидания ресурса исполняющей системы;
- 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 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

Приоритет потока повышается относительно базового приоритета. На рисунке 2.1 показано, что после повышения приоритета поток выполняется с повышенным приоритетом в течение одного кванта времени, а затем приоритет понижается на 1 уровень с каждым последующим квантом. Цикл продолжается до тех пор, пока приоритет не понизится до базового.

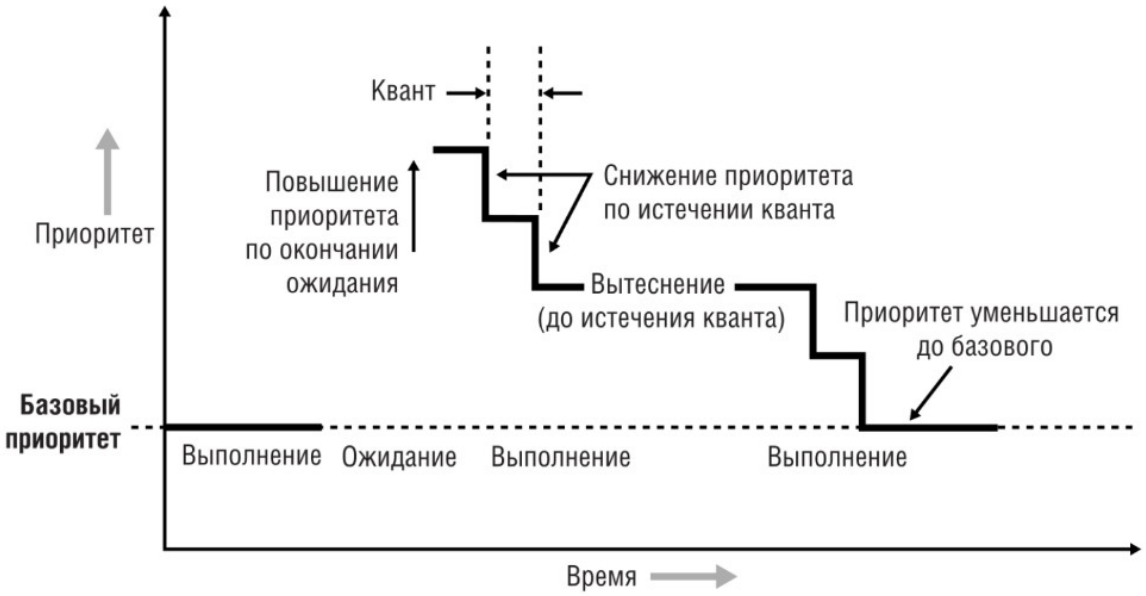

Рисунок 2.1 – Динамическое изменение приоритета

2.3.2 MMCSS

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

Одно из наиболее важных свойств для планирования потоков – категория планирования (Sheduling Category) – первичный фактор определяющий приоритет потоков, зарегистрированных в MMCSS. Различные категории планирования представленны в таблице 2.4.

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

Функции MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, соответствующего их категориям планирования. Затем их приоритет понижается до уровня, соответствующего категории Exhausted с целью предоставления возможности получения ресурса другими потоками.

Вывод

Функции обработчика прерывания от системного таймера в системах `Unix` и `Windows` решают схожие задачи:

- декремент счетчиков времени:
  - часов
  - таймеров
  - счетчиков времени отложенных действий
  - будильников реального времени;
- декремент кванта текущего процесса в `UNIX` и декремент текущего потока в `Windows`;
- инициализация отложенных действий, относящихся к работе планировщика, например, пересчёт приоритетов.

Данная схожесть обусловлена тем, что и `UNIX` и `Windows` являются системами разделения времени с динамическим расчетом приоритетов и вытеснением. Тем не менее, данные системы различаются своим подходом к планированию и пересчету приоритетов как процессов, так и потоков.