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»

Тема Прерывание таймера в Windows и UNIX

Студент Богаченко А. Е.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Москва — 2020 г.

1 Функции обработчика прерываний от системного таймера

В большинстве систем UNIX определенно понятие основного тика, который равен N тикам таймера (число N зависит от конкретного варианта системы).

В каждой `Unix`-машине есть аппаратный таймер, который вырабатывает прерывание в системе через определённые промежутки времени. Период времени между двумя такими прерываниями (тиками) в ОС `Unix` равен 10 мс. Обработчик прерывания по таймеру является вторым по приоритету – первым является обработчик прерывания по сбою питания.

Тик – период времени между двумя последующими прерываниями таймера.

Основной тик – период времени равный N тикам таймера, где N – системозависимая величина.

Квант – период времени, отведенный планировщиком процессу для выполнения.

1.1 ОС семейства Unix/Linux

По тику:

- инкремент счётчика тиков аппаратного таймера;
- обновление часов и других таймеров системы;
- обновление статистики использования процессора текущим процессом: инкремент поля `p_cpu` дескриптора текущего процесса на единицу, до максимального значения, равного 127;
- декремент счётчика времени до отправления на выполнение отложенных вызов. Если счётчик достиг нуля, то выставление флага для обработчика отложенных вызов;

- декремент кванта текущего потока.

По главному тику:

- инициализация отложенных вызовов функций, относящихся к работе планировщика, таких как пересчёт приоритетов;
- выход из состояния прерываемого сна системных процессов `swapper` и `pagedaemon`.
- декремент счётчиков времени, отображающих оставшееся до посылки одного из сигналов тревоги время:
  - `SIGALARM` — сигнал будильника реального времени, который отправляется по истечении заданного промежутка реального времени;
  - `SIGPROF` — сигнал будильника профиля процесса, который измеряет время работы процесса;
  - `SIGVTALRM` — сигнал будильника виртуального времени, который измеряет время работы процесса в режиме задачи.

По кванту:

- при превышении текущим процессом выделенного кванта, посылка сигнала `SIGXCPU` – превышение лимита процессорного времени – этому процессу.

1.2 Windows-системы

Всего в ОС Windows 32 уровня запроса прерывания (от 0 до 31). Прерывания обслуживаются в порядке их приоритета. У интервального таймера системных часов высокое значение `IRQL` – `CLOCK_LEVEL`):

По тику:

- инкремент счётчика системного времени;

- декремент 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;

- декремент счётчиков времени отложенных задач;

- если активен механизм профилирования ядра, то инициализация отложенного вызова обработчика ловушки профилирования ядра путём постановки объекта в очередь `DPC`: обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

По главному тику:

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

По кванту:

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

2 Пересчёт динамических приоритетов

2.1 ОС семейства Unix/Linux

В ОС семейства `Unix/Linux` динамически пересчитываться могут только приоритеты пользовательских процессов.

Очередь готовых к выполнению процессов формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования: 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.

В современных системах `Unix` ядро является вытесняющим – процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра.

Дескриптор процесса proc содержит следующие поля, относящиеся к приоритету:

2.1.1 Приоритеты процессов

Приоритет процесса в `UNIX` задаётся числом в диапазоне от 0 до 127, причём чем меньше значение, тем выше приоритет. Приоритеты 0–49 зарезервированы ядром операционной системы, прикладные процессы могут обладать приоритетом в диапазоне от 50 до 127. Приоритеты ядра являются фиксированными величинами.

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

- фактор любезности – целое число в диапазоне от 0 до 39. Чем меньше значение фактора любезности, тем выше приоритет процесса. Фактор любезности процесса может быть изменён суперпользователем системным вызовом `nice`;

- степень загруженности процессора в момент последнего обслуживания им процесса.

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

- `p_pri` – текущий приоритет планирования;
- `p_usrpri` – приоритет режима задачи;
- `p_cpu` – результат последнего измерения использования процессора;
- `p_nice` – показатель уступчивости, устанавливаемый пользователем.

Планировщик использует поле `p_pri` для принятия решения о том, какой процесс отправить на выполнение. Значения `p_pri` и `p_usrpri` одинаковы, когда процесс находится в режиме задачи. Когда процесс просыпается после блокировки в системном вызове, его приоритет временно повышается. Планировщик использует `p_usrpri` для хранения приоритета, который будет назначен процессу при переходе из режима ядра в режим задачи, а `p_pri` – для хранения временного приоритета для выполнения в режиме ядра.

Ядро связывает приоритет сна (0–49) с событием или ожидаемым ресурсом, из-за которого процесс может быть заблокирован. Когда блокированный процесс просыпается, ядро устанавливает `p_pri`, равное приоритету сна события или ресурса, на котором он был заблокирован, следовательно, такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи.

В таблице 2.1 приведены значения приоритетов сна для систем `4.3BSD UNIX` и `SCO UNIX`. Такой подход позволяет системным вызовам быстрее завершать свою работу. По завершении процессом системного вызова его приоритет сбрасывается в значение текущего приоритета в режиме задачи. Если при этом приоритет окажется ниже, чем приоритет другого запущенного процесса, ядро произведет переключение контекста.

Приоритет в режиме задачи зависит от уступчивости и последней измеренной величины использования процессора. Степень уступчивости – это число в диапазоне от 0 до 39 со значением 20 по умолчанию.

Таблица 2.1 – Системные приоритеты сна

Событие	**Приоритет 4.3BSD UNIX**	**Приоритет SCO UNIX**
Ожидание загрузки в память страницы	0	95
Ожидание индексного дескриптора	10	88
Ожидание ввода–вывода	20	81
Ожидание буфера	30	80
Ожидание терминального ввода	30	75
Ожидание терминального вывода	30	74
Ожидание завершения выполнения	30	73
Ожидание события	40	66

Системы разделения времени стараются выделить процессорное время таким образом, чтобы все процессы системы получили его в равных количествах, что требует слежения за использованием процессора. Поле `p_cpu` содержит величину последнего измерения использования процессора процессом. При создании процесса это поле инициализируется нулем. На каждом тике обработчик таймера увеличивает `p_cpu` на единицу для текущего процесса, вплоть до максимального значения – 127. Каждую секунду ядро вызывает процедуру `schedcpu`, которая уменьшает значение `p_cpu` каждого процесса исходя из фактора «полураспада».

В `4.3BSD` для расчета применяется формула

$$d = \frac{2 \cdot la}{2 \cdot la + 1},$$

где `la` – `load_average` – это среднее количество процессов в состоянии готовности за последнюю секунду.

Кроме того, процедура `schedcpu` также пересчитывает приоритеты режима задачи всех процессов по формуле

$$p_{usrpri} = PUSER + \frac{p\_cpu}{4} + 2 \cdot p\_nice,$$

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

Если процесс до вытеснения другим процессом использовал большое

количество процессорного времени, его `p_cpu` будет увеличен, что приведет к увеличению значения `p_usrpri` и к понижению приоритета.

Чем дольше процесс простаивает в очереди на выполнение, тем меньше его `p_cpu`. Это позволяет предотвратить зависания низкоприоритетных процессов. Если процесс бóльшую часть времени выполнения тратит на ожидание ввода-вывода, то он остается с высоким приоритетом.

Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы получили его примерно в равных количествах. Фактор полураспада обеспечивает экспоненциально взвешанное среднее значение использования процессора в течение функционирования процесса. Формула, применяемая в SVR3 имеет недостаток: вычисляя простое экспоненциальное среднее, она способствует росту приоритетов при увеличении загрузки системы.

2.2 Windows-системы

В системе 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 потока не истёк.

В `Windows` за планирование отвечает совокупность процедур ядра, называемая диспетчером ядра. Диспетчеризация может быть вызвана, если:

- поток готов к выполнению;
- истёк квант текущего потока;
- поток завершается или переходит в состояние ожидания;
- изменился приоритет потока;
- изменилась привязка потока к процессору.

2.2.1 Приоритеты потоков

В системе предусмотрено 32 уровня приоритетов: уровни реального времени (16–31), динамические уровни (1–15) и системный уровень (0).

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)).

Относительный приоритет – это приращение к базовому приоритету процесса.

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

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

С точки зрения планировщика `Windows` важно только значение приоритета. Процесс обладает только базовым приоритетом, тогда как поток имеет базовый, который наследуется от приоритета процесса, и текущий приоритет. Операционная система может на короткие интервалы времени повышать приоритеты потоков из динамического диапазона, но никогда не регулирует приоритеты потоков в диапазоне реального времени. Приложения пользователя обычно запускаются с базовым приоритетом *normal*. Некоторые системные процессы имеют приоритет выше 8, что гарантирует, что потоки в этих процессах будут запускаться с более высоким приоритетом.

Система динамически повышает приоритет текущего потока в следующих случаях:

- по завершении операции ввода-вывода;
- по окончании ожидания на событии или семафоре исполнительной системы;
- по окончании ожидания потоками активного процесса;
- при пробуждении `GUI`-потоков из-за операции с окнами;

- если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени.

Динамическое повышение приоритета применяется только к потокам из динамического диапазона (1–15). Приоритет потока не может оказаться выше 15.

Рассмотрим каждый случай в отдельности.

2.2.2 Повышение приоритета по завершении операции ввода-вывода

По окончании определенных операций ввода-вывода Windows временно повышает приоритет потоков и потоки, ожидающие завершения этих операций, имеют больше шансов возобновить выполнение и обработать полученные от устройств ввода-вывода данные.

Драйвер устройства ввода-вывода через функцию `IoCompleteRequest` указывает на необходимость динамического повышения приоритета после выполнения соответствующего запроса.

В таблице 2.3 приведены приращения приоритетов.

Таблица 2.3 – Рекомендованные приращения 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 в течение одного кванта выполняется с повышенным приоритетом, а затем приоритет снижается на один уровень с каждым последующим квантом. Цикл продолжается до тех пор, пока приоритет не снизится до базового.

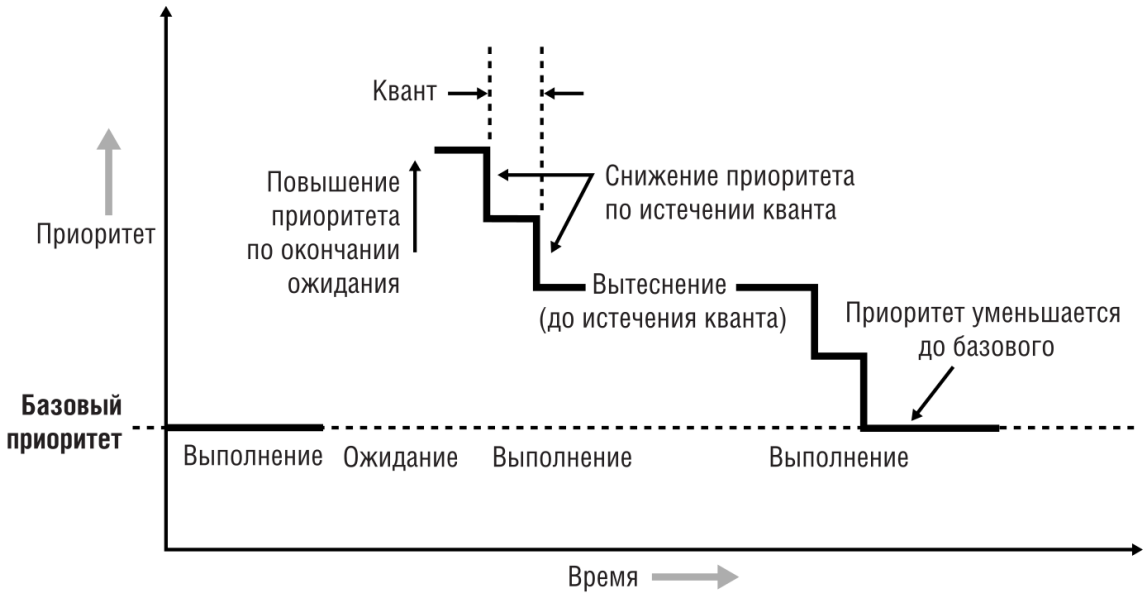

Рисунок 2.1 – Динамическое изменение приоритета

2.2.3 Повышение приоритета по окончании ожидания на событии или семафоре

Если ожидание потока на событии системы или семафоре успешно завершается из-за вы `SetEvent`, `PulseEvent` или `ReleaseSemaphore`, его приоритет повышается на 1. Такая регулировка позволяет равномернее распределить процессорное время – потокам, блокируемым на событиях, процессорное время требуется реже, чем остальным. В данном случае действуют те же правила динамического повышения приоритета.

К потокам, пробуждающимся в результате установки события вызовом функций `NtSetEventBoostPriority` и `KeSetEventBoostPriority`, повышение приоритета применяется особым образом.

2.2.4 Повышение приоритета по окончании ожидания потоками активного процесса

Если поток в активном процессе завершает ожидание на объекте ядра, функция ядра `KiUnwaitThread` повышает его текущий приоритет на величину значения `PsPrioritySeparation`. `PsPrioritySeparation` – это индекс в таблице квантов, с помощью которой выбираются величины квантов

для потоков активных процессов. Какой процесс является в данный момент активным, определяет подсистема управления окнами.

Приоритет повышается для создания преимуществ интерактивным приложениям по окончании ожидания, в результате чего повышаются шансы на возобновление потока приложения. Важной особенностью данного вида динамического повышения приоритета является то, что он поддерживается всеми системами Windows и не может быть отключен даже функцией `SetThreadPriorityBoost`.

2.2.5 Повышение приоритета при пробуждении GUI-потоков

Приоритет потоков окон пользовательского интерфейса повышается на 2 после их пробуждения из-за активности подсистемы управления окнами. Приоритет повышается по той же причине, что и в предыдущем случае, – для увеличения отзывчивости интерактивных приложений.

2.2.6 Повышение приоритета при нехватке процессорного времени

Раз в секунду диспетчер настройки баланса – системный поток, предназначенный для выполнения функций управления памятью – сканирует очереди готовых потоков и ищет потоки, которые находятся в состоянии готовности в течение примерно 4 секунд. Диспетчер настройки баланса повышает приоритет таких потоков до 15. Причем в `Windows 2000` и `Windows XP` квант потока удваивается относительно кванта процесса, а в `Windows Server 2003` квант устанавливается равным 4 единицам. По истечении кванта приоритет потока снижается до исходного уровня. Если потоку все еще не хватило процессорного времени, то после снижения приоритета он возвращается в очередь готовых процессов. Через 4 секунды он может снова получить повышение приоритета.

Чтобы свести к минимуму расход процессорного времени, диспетчер настройки баланса сканирует только 16 готовых потоков за раз, а повышает

приоритет не более чем у 10 потоков за раз. Диспетчер настройки баланса не решает всех проблем с приоритетами потоков, однако позволяет потокам, которым не хватает процессорного времени, получить его.

Вывод

Операционные системы UNIX и Windows являются системами разделения времени с вытеснением. В связи с этим обработчики прерываний от системных таймеров в них выполняют схожие функции:

- инициализируют отложенные действия, относящиеся к работе планировщика, такие как пересчёт приоритетов;
- выполняют декремент счётчиков времени: часов, таймеров, будильников реального времени, счётчиков времени отложенных действий;
- выполняют декремент кванта: текущего процесса в Linux, текущего потока в Windows.

Такую схожесть можно объяснить тем, что обе системы являются системами разделения времени с вытеснением и динамическими приоритетами, но несмотря на это, системы планирования в ОС семейства Windows и семейства Unix/Linux различаются:

- при создании процесса в Windows, ему назначается базовый приоритет. Приоритеты потоков определяются относительно приоритета процесса, в котором они создаются. Приоритет потока пользовательского процесса может быть пересчитан динамически;
- в Unix/Linux приоритет процесса характеризуется текущим приоритетом и приоритетом процесса в режиме задачи. Приоритет пользовательского процесса может быть динамически пересчитан.