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
(вторая часть)

Дисциплина: Операционные системы

Тема: Функции обработчика прерывания от системного таймера в защищенном режиме. Пересчет динамических приоритетов.

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Рязанова Н. Ю.

Москва, 2020 г.

Оглавление

1. Функции системного таймера в защищенном режиме.

1.1. Windows.

По тику.

- Инкремент счетчика системного времени.
- Декремент счетчиков отложенных задач.
- Декремент остатка кванта текущего потока.

По главному тику.

- Инициализация диспетчера настройки баланса (освобождение объекта «событие» каждую секунду).

По кванту.

- Инициализация диспетчеризации потоков (добавление соответствующего объекта DPC в очередь).

1.2. Unix/Linux.

По тику.

- Инкремент счетчика тиков аппаратного таймера.
- Инкремент счетчика использования процессора текущими процессом.
- Инкремент часов и других таймеров системы.
- Декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов (при достижении нулевого значения, установка флага, указывающего на необходимость запуска обработчика отложенного вызова).

По главному тику.

- Инициализация отложенных вызовов функций, относящихся к работе планировщика, путем посылки соответствующего сигнала или изменения состояния процесса с S на R.
- Пробуждение системных процессов, таких как swapper и pagedaemon (перемещение дескрипторов процессов из очереди «спящих» в очередь «готовых к выполнению»).

- Декремент счетчиков времени, оставшегося до отправления одного из сигналов:

  SIGALARM (декремент будильника реального времени)

  SIGPROF (измерение времени работы процесса)

  SIGVTALARM (измерение времени работы процесса в режиме задачи).

По кванту.

- Посылка текущему процессу сигнала SIGXCPU, если израсходован выделенный ему квант процессорного времени.

2. Пересчет динамических приоритетов.

Пересчет приоритетов возможен только для пользовательских процессов.

2.1. Windows.

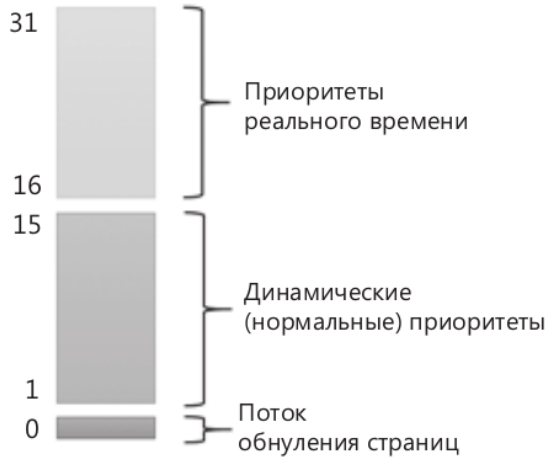
При создании процесса ему назначается приоритет. В Windows он называется базовым.

Уровни приоритета

В системе Windows существует 32 уровня приоритета, от 0 до 31 (31 – наивысший):

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

Рисунок 2.1. Уровни приоритетов.



Уровни приоритета назначаются Windows API и ядром Windows.

I. Базовый приоритет. Windows API систематизирует процессы по классу приоритета, который им назначается при создании:

реального времени	Real-time (4)
высокий	High (3)
выше обычного	Above Normal (6)
обычный	Normal (2)
ниже обычного	Below Normal (5)
уровень простоя	Idle (1)

II. Текущий приоритет. Внутри процессов назначается относительный приоритет потока, который является приращением к базовому приоритету процесса.

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)
уровень простоя	Idle (-15)

Таким образом, каждый процесс имеет только *базовое* значение приоритета, которое назначается ему при создании. Каждый поток имеет два значения приоритета: *базовое* (наследуется от приоритета процесса) и *текущее*.

Таблица 2.1. Соответствие между приоритетами ядра Windows и приоритетами Windows API

Класс приоритета/относительный приоритет	**Real-time**	**High**	**Above-Normal**	**Normal**	**Below-Normal**	**Idle**
Time Critical (+ насыщение)	31	15	15	15	15	15
Highest (+2)	26	15	12	10	8	6
Above Normal (+1)	25	14	11	9	7	5
Normal (0)	24	13	10	8	6	4
Below Normal (-1)	23	12	9	7	5	3
Lowest (-2)	22	11	8	6	4	2
Idle (–Saturation)	16	1	1	1	1	1

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом. При этом текущий выполняемый поток с более низким приоритетом может быть вытеснен еще до окончания его кванта времени.

ОС Windows никогда не повышает приоритет потоков в диапазоне реального времени. Динамическое повышение приоритета потока в Windows применяется только для потоков с приоритетом динамического диапазона (0-15). Однако, независимо от величины приращения, приоритет потока не превышает 15.

Случаи повышения текущих приоритетов потока.

1. *Завершение ввода/вывода.*

   При завершении определенных операций ввода/вывода у потоков, ожидавших завершения этих операций, больше шансов немедленно возобновить выполнение. Важно, что для запросов на ввод/вывод, адресованных устройствам с меньшим временем отклика, предусматриваются большие приращения приоритета.

Рисунок 2.2. Рекомендуемые значения повышения приоритета.

Устройство	**Повышение**
Диск, CD-ROM, параллельный порт, видео	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковая карта	8

2. *Завершение ожидания на мьютексе, семафоре или другом событии.*

   Поток получает повышение приоритета на 2 уровня, если он находится в фоновом процессе, и на 1 уровень во всех остальных случаях. Такое повышение позволяет равномернее распределять процессорное время (потокам, блокируемым на событиях, процессорное время требуется реже, чем остальным).

3. *Завершение ожидания потоками активного процесса.*

   Когда поток в активном процессе завершает ожидание на объекте ядра, ядро повышает его текущий приоритет на текущее значение переменной PsPrioritySeparation. Повышение приоритета необходимо для улучшения скорости отклика интерактивных приложений.

4. *Пробуждение GUI-потока.*

   Повышения приоритетов потоков-владельцев окон на 2 требуется из-за активности подсистемы управления окнами. Повышение приоритета необходимо для увеличения отзывчивости интерактивного приложения по окончании ожидания и повышения шансов на немедленное возобновление его потока.

5. *Задержка готового процесса из-за нехватки процессорного времени.*

   Для преодоления инверсий приоритетов, требуется отслеживать блокировки вместе с их владельцами и повышение приоритетов для продвижения работы. Каждую секунду диспетчер настройки баланса (системный поток) сканирует очередь готовых потоков в поисках потоков, ожидающих выполнения около 4 секунд. У найденных потоков диспетчер настройки баланса повышает приоритет до максимального значения – 15, и устанавливается квант времени. По истечении установленного кванта времени, приоритет потока понижается до исходного уровня. Если поток не успел завершить свою работу или был вытеснен потоком с более высоким приоритетом, то он будет возвращен в очередь готовых потоков. Чтобы минимизировать расход процессорного времени, диспетчер настройки баланса сканирует лишь 16 готовых потоков. Кроме того, он повышает приоритет не более, чем у 10 потоков за один проход. Обнаружив 10 потоков, приоритет которых следует повысить (что говорит о высокой загруженности системы), он прекращает сканирование. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз.

Уровни запросов прерываний.

Для обеспечения поддержки мультизадачности системы, когда исполняется код режима ядра, Windows использует приоритеты прерываний IRQL.

Внутри ядра IRQL представляются в виде номеров от 0 до 31 для системы x86. Ядро определяет стандартный набор IRQL для программных прерываний, а HAL связывает IRQL с номерами аппаратных прерываний.

Прерывания обслуживаются в порядке их приоритета. При возникновении прерывания с высоким приоритетом процессор сохраняет информацию о состоянии прерванного потока и активизирует сопоставленный с данным прерыванием диспетчер ловушки. Последний повышает IRQL и вызывает процедуру обслуживания прерывания (ISR). После выполнения ISR прерванный поток возобновляется с той точки, где он был прерван.

2.2. Unix/Linux.

Основной задачей планировщика является принятие решения о том, когда произвести переключение контекста и какой из процессов назначить на выполнение. Механизм планирования в традиционных системах базируется на приоритетах.

Каждый процесс обладает приоритетом планирования, изменяющимся с течением времени. Планировщик всегда выбирает процессы, обладающие более высоким приоритетом. Приоритет процесса не является фиксированным и динамически изменяется системой в зависимости от использования вычислительных ресурсов, времени ожидания запуска и текущего состояния процесса.

Традиционное ядро UNIX является строго невытесняющим. Выполняющийся процесс может только добровольно освободить процессор в случае своего блокирования в ожидании ресурса, иначе он может быть вытеснен при переходе в режим задачи. Такая реализация решает проблемы синхронизации, связанные с доступом нескольких процессов к одним и тем же ресурсам.

Современное ядро Linux (начиная с версии 2.5) является полностью вытесняемым, поскольку оно должно обеспечивать работу процессов реального времени.

Приоритет процесса задается любым целым числом, лежащим в диапазоне от 0 до 127 (0 – наивысший уровень). Приоритеты 0-49 зарезервированы для ядра, приоритеты 50-127 могут назначаться прикладным процессам.

Структура proc содержит следующие поля, относящиеся к приоритетам:

p_pri	Текущий приоритет планирования	Используется планировщиком для принятия решения о том, какой процесс направить на выполнение.
p_usrpri	Приоритет режима задачи	Используется планировщиком для хранения приоритета, назначенного после возвращения в режим задачи.
p_cpu	Результат последнего измерения использования процессора	Используется для предотвращения зависания низкоприоритетных процессов по вине операционной системы.
p_nice	Фактор «любезности», устанавливаемый пользователем	Используется для изменения уровня приоритета. Увеличение значение приводит к уменьшению приоритета.

Когда процесс просыпается, ядро устанавливает p_pri, равное значению приоритета сна события или ресурса, на котором он был заблокирован. Такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи.

Когда процесс завершил выполнение системного вызова и находится в состоянии возврата в режим задачи, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи.

Пересчет приоритетов для режима задачи всех процессов производится по формуле (1).

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2\,p\_nice, \qquad (1)$$

где PUSER – базовый приоритет в режиме задачи.

Так, приоритет в режиме задачи зависит от следующих факторов:

1. количество процессорного времени, использованного в последний раз (p_cpu);
2. степень «любезности» (p_nice).

1. Поле p_cpu содержит величину результата последнего сделанного измерения использования процессора процессом. Эта величина увеличивается на 1 для текущего процесса на каждом тике обработчиком таймера до максимального значения 127. Также, она уменьшается исходя из фактора «полураспада» для каждого процесса каждую секунду процедурой schedcpu. Для расчета фактора полураспада применяется формула (2).

$$decay = \frac{2 * load\_average}{2 * load\_average + 1} \qquad (2)$$

Фактор полураспада обеспечивает экспотенциально взвешенное среднее значение использования процессора в течение всего периода функционирования процесса.

2. Степень любезности – число в диапазоне от 0 до 39 со значением 20 по умолчанию. Увеличение значения приводит к уменьшению приоритета. Уменьшить эту величину может только суперпользователь, поскольку при этом поднимется его приоритет. Одни пользователи могут быть поставлены в более выгодные условия другими пользователями посредством

увеличения кем-либо из последних значения уровня любезности для своих менее важных процессов.

Вывод

Операционные системы Unix/Linux и Windows являются системами разделения времени с динамическими приоритетами и вытеснением. Отсюда следует наличие схожих функций системного таймера: инкремент счетчика системного времени, декремент кванта, инициализация отложенных действий, относящихся к работе планировщика, декремент счетчиков времени, оставшегося до выполнения отложенных вызовов.

Однако системы планирования в операционных системах семейства Windows и семейства Unix/Linux различны. Классический Unix имел невытесняющее ядро, в то время как Windows и современные Unix/Linux являются полностью вытесняемыми (вытесняемость современных ядер обусловлена необходимостью поддержки процессов реального времени, таких как аудио, видео).