Операционные системы

Лабораторная работа №1, часть 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | Студент | Юмаев А.Р. |
|  |  |  | Группа | ИУ7-55 |
|  |  |  | Преподаватель | Рязанова Н.Ю. |

Оглавление

[Введение 2](#_Toc21881047)

[1. Функции прерывания системного таймера 3](#_Toc21881048)

[1.1 Windows/DOS 3](#_Toc21881049)

[1.2 Unix/Linux 4](#_Toc21881050)

[2. Пересчета динамических приоритетов для ОС семейства Windows и Unix/Linux 5](#_Toc21881051)

[2.1 Windows 5](#_Toc21881052)

[2.2 Unix/Linux 7](#_Toc21881053)

[2.2.1 Фактор полураспада 9](#_Toc21881054)

[Вывод 9](#_Toc21881055)

# Введение

Для аппаратной платформы x86 главный системный таймер — это программируемый интервальный таймер (programmable interval timer, PIT). Таймер PIT существует на всех машинах платформы PC. Co времен операционной системы DOS он используется для управления прерываниями. Идея, которая лежит в основе системного таймера — это обеспечение механизма управления прерываниями, которые возникают периодически с постоянной частотой.

Системный таймер в Windows имеет большое значение для функционирования компьютера. От него зависит отслеживание системного времени, работа планировщиков задач и выдача квантов времени процессам

Большинство систем вводят нотацию *главного тика* (major tick), который происходит каждые n тиков, где n зависит от конкретной версии системы. Определенный набор функций выполняется только на главных тиках. Например, 4.3BSD производит пересчет приоритетов каждые 4 тика, a SVR4 обрабатывает алармы (alarm) и производит пробуждение системных процессов раз в секунду.

 Обработчик прерываний (или процедура обслуживания прерываний) — специальная процедура, вызываемая по прерыванию для выполнения его обработки. Обработчики прерываний могут выполнять множество функций, которые зависят от причины, которая вызвала прерывание.

Когда происходит прерывание, управление передается собственно обработчику прерываний по определенном адресу. Этот адрес находится в специальной таблице, называемой таблицей **векторов прерываний**. Вектор прерывания представляет собой пару ***сегмент:смещение***, то есть адрес собственно функции - обработчика прерывания. Перед тем как передать управление обработчику прерываний, процессор сохраняет в стеке адрес текущей команды (пару CS:IP) и регистр флагов. После того, как обработчик прерываний закончил выполняться, процессор восстанавливает из стека пару CS:IP и передает управление по этому адресу. Также восстанавливается регистр флагов.

Таблица векторов прерываний имеет следующий вид. Вектор прерывания представляет собой пару сегмент/смещение, то есть 4 байта. Таблица прерываний физически располагается с адреса 0000h и заканчивая адресом 03FFh. Прерыванию INT 0h соответствует вектор, расположенный по адресам 0-3, прерыванию INT 1h - соответственно 4-7, т.е. адрес обработчика прерывания N находится по адресам, начиная с N\*4 до N\*4+3. Например, обработчик прерывания INT 21h (вызов подсистемы DOS) будет расположен по физическому адресу 0000h:0084h. Всего в таблице прерываний может поместиться 256 векторов.

Вектор прерываний имеет следующую структуру: в младшей паре находится значение IP (фактически, смещение), в старшей - значение сегмента команд (CS), сначала идет смещение, а потом - сегмент.

Импульсы, поступающие с выхода канала 0 таймера, вызывают прерывание 8. Обработчик этого прерывания в BIOS подсчитывает количество таких импульсов в 4-байтной области памяти (два 2- байтных слова). Этот счетчик, находящийся в области памяти BIOS по адресу 0040:006C, таким образом, хранит количество тиков таймера, прошедших от полуночи (0 в счетчике соответствует полночи). При запуске системы BIOS запрашивает у оператора время дня, переводит его в количество тиков и записывает по указанному адресу. Затем в процессе работы это число модифицируется обработчиком прерывания 8. То обстоятельство, что обработчик прерывания 8 в BIOS обеспечивает работу службы времени, следует учитывать при перехвате прерывания 8 и при перепрограммировании канала 0 таймера.

Доступ к счетчику времени поддерживается прерыванием 0x1A. При обращении к этому прерыванию со значением 0 в регистре AH мы получаем в CX старшую, а в DX - младшую части счетчика. При обращении со значением 1 в AH мы задаем счетчик в регистрах CX, DX, и это значение записывается в память BIOS.

Квант — это продолжительность времени, в течение которого потоку разрешено работать, пока не настанет очередь запускаться другому потоку с тем же уровнем приоритета. Значение кванта может варьироваться от системы к системе и от процессора к процессору. Обычно время стандартное время кванта записано в специальной переменной в памяти и составляет около 10мс.

# 1. Функции прерывания системного таймера

## 1.1 Windows/DOS

По кванту

* Постановка в очередь DPC Queue программного прерывания DPC (отложенный вызов процедур для минимизации времени работы обработчика прерывания)
* Перепланирование времени использования процессора после истечения кванта
* Подсчет тактовых циклов для отслеживания времени выполнения потока по формуле: (частота процессора в Гц, которая хранится в PRCB-поле) \* (количество секунд, затрачиваемое на запуск одного такта системных часов). Так как время выполнения потока вычисляется на основе циклов процессора
* Если есть еще один поток, имеющий такой же приоритет – происходит переключение контекста на следующий поток в очереди готовых потоков. Если поток исчерпал свой квант, а других потоков с его уровнем приоритета нет, Windows позволяет потоку выполняться в течение еще одного кванта времени
* Переключение контекста

По основному тику (major tick, в Windows происходит раз в секунду)

* Сканирование очереди из 16 готовых потоков в поиске тех из них, которые находятся в состоянии ожидания (то есть не были запущены) около 4 секунд. Если такой поток будет найден, диспетчер настройки баланса (*balance set manager*) повышает его приоритет до 15 единиц и устанавливает квантовую цель эквивалентной тактовой частоте процессора при подсчете 3 квантовых единиц.

По тику (такту)

* Контроль за работой двигателей НГМД
* Вызов прерывания INT 1Ch
* Возникает прерывание на IRQL-уровне Clock
  + Декремент остатка кванта времени у текущего потока
  + Увеличение счетчика тиков таймера (uptime, time of day)

## 1.2 Unix/Linux

По кванту

* Переключение контекста
* Отправка текущему процессу сигнала SIGXCPU, если тот превысил выделенный квант

По основному тику (каждый четвертый тик)

* Пересчет приоритетов процессов планировщиком
* Декременты времени сигналов SIGALARM, SIGPROF, SIGVTALARM

Каждую секунду

* Вызов процедуры schedcpu() (запускаемую через отложенный вызов), которая уменьшает значение, которая уменьшает значение p\_cpu для каждого процесса, исходя из фактора “полураспада”.

По тику (начальное значение - 10 мс хранится в константе HZ в файле param.h)

* Обновление статистики по использованию процессорного времени и других ресурсов
* Обновление значения времени работы системы (uptime), абсолютного времени (time of day)
* Подсчет тиков аппаратного таймера
* Пробуждение системных процессов *swapper* и *pagedaemon*
* Обработка отложенных вызовов
* Увеличение p\_cpu на 1 для текущего процесса

# 2. Пересчета динамических приоритетов для ОС семейства Windows и Unix/Linux

Планирование процессов в ОС Windows и Unix/Linux – это определение приоритета процесса и постановка его в очередь на выполнение к процессору. Диспетчеризация – выделение процессу процессорного времени. Приоритет процесса не является фиксированным и может изменяться в процессе его выполнения. Процессор может остановить текущий процесс A, если готов к запуску процесс B, который имеет больший, чем А, приоритет, даже если процесс А не израсходовал свой квант.

## 2.1 Windows

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом. Windows осуществляет планировку потоков по вытесняющему принципу. Такой подход имеет смысл, если учесть, что процессы не запускаются, а только предоставляют ресурсы и контекст, в котором запускаются потоки.

Внутренне Windows использует 32 уровня приоритета, от 0 до 31.

* 16 уровней реального времени (от 16 до 31)
* 16 уровней изменяющихся уровней (от 0 до 15), и которых 0 зарезервирован для потока обнуления страниц

Уровни приоритета потоков назначаются исходя из двух разных позиций: одной от Windows API и другой от ядра Windows. Сначала Windows API систематизирует процессы по классу приоритета, который им присваивается при создании (номера представляют внутренний индекс PROCESS\_PRIORITY\_CLASS, распознаваемый ядром):

* Реального времени — Real-time (4)
* Высокий — High (3)
* Выше обычного — Above Normal (7)
* Обычный — Normal (2)
* Ниже обычного — Below Normal (5)
* Простоя — Idle (1)

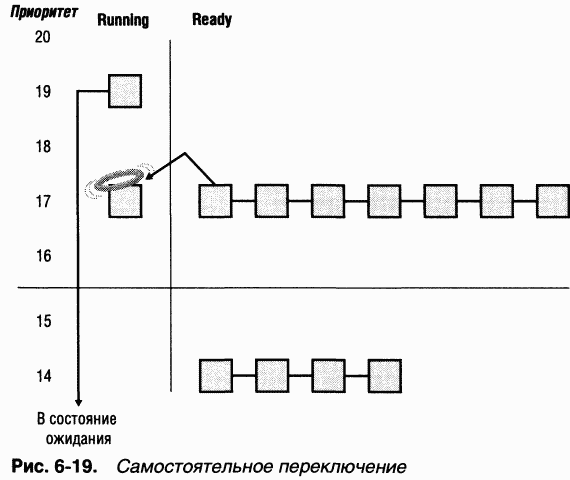
Затем назначается относительный приоритет отдельных потоков внутри этих процессов. Здесь номера представляют изменение приоритета, применяющееся к базовому приоритету процесса:

* 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)
* Простоя — Idle (–15)

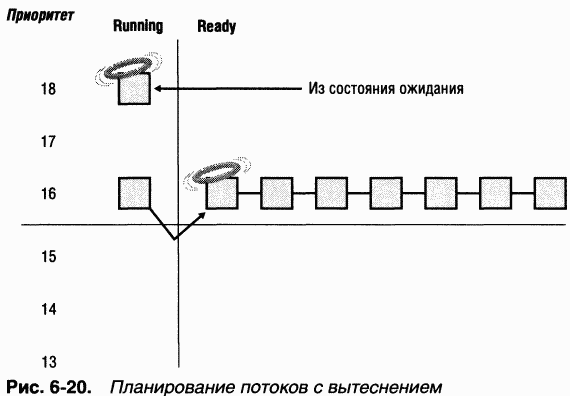
Исходный базовый приоритет потока наследуется из базового приоритета процесса. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал. Изменение приоритета процесса изменяют приоритеты потоков, повышая их или снижая, но их относительные установки остаются прежними.

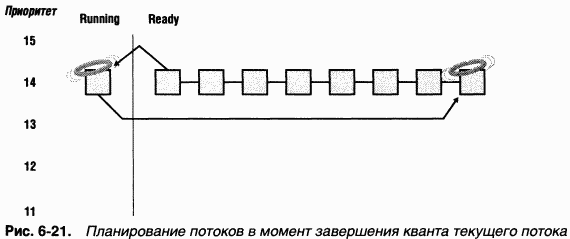
Сценарии планирования:

* Самостоятельное переключение (переход в ожидание вызовом функции)



* Вытеснение (текущий поток вытеснен потоком с более высоким приоритетов, в независимости от режима – ядра или пользовательского)



* Завершение кванта (понижение приоритета и выбор потока из очереди готовых к выполнению)  
    
  
* Переключение контекста
* Поток простоя (если нет ни одного потока, готового к выполнению на процессоре, Windows подключает к данному процессору поток простоя)
* Динамическое повышение приоритета. Windows может динамически повышать значение текущего приоритета потока в пяти случаях:
  + После завершения операций ввода-вывода
  + По окончании ожидания на событии или семафоре исполнительной системы
  + По окончании операции ожидания потоками активного процесса
  + При пробуждении GUI-потоков из-за операций с окнами
  + Если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени

## 2.2 Unix/Linux

Unix изначально создавался как ОС разделения времени, что означает, что одновременно может выполняться несколько процессов. На самом низком уровне планировщик заставляет процессор переключаться от одного процесса к другому, это называется смена контекста. Контекст – моментальный снимок состояний регистров общего назначения, управления памятью, и других регистров процессора. После сохранения регистров в стек, ядро загружает контекст нового процесса и начинает его выполнение. Цель политики планирования заключается в увеличении скорости реакции при интерактивном взаимодействии пользователя с системой. Для диспетчеризации процессов с равным приоритетом применяется вытесняющее квантование времени, но традиционное ядро Unix является строго не вытесняющим.

Приоритет процесса в Unix задается числом от 0 до 127. Чем меньше число, тем больше приоритет. Приоритеты от 0 до 49 зарезервированы для ядра. Пользовательские процессы могут иметь приоритеты от 50 до 127.

Структура proc имеет следующие поля, относящиеся к приоритетам:

|  |  |
| --- | --- |
| p\_pri | Текущий приоритет планирования  Временный приоритет в режиме ядра |
| p\_usrpri | Приоритет режима задачи |
| p\_cpu | Результат последнего измерения использования процессора |
| p\_nice | Фактор любезности, устанавливается пользователем |

Планировщик используется поле p\_pri для того, чтобы понять какой процесс направить на выполнение. Когда процесс просыпается после блокировки в системном вызове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра.

Приоритет в режиме задачи зависит от двух факторов: любезности nice и последней измеренной величины использования процессора. Степень любезности является числом от 0 до 39 со значением 20 по умолчанию. Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы получили его примерно в равном количестве.

В системе SVR3 используется фиксированное значение фактора полураспада – ½. В 4.3BSD для расчета фактора полураспада применяется формула:

decay = (2 \* load\_average) / (2 \* load\_average + 1),

где load\_average – среднее количество процессов,

находящихся в состоянии готовности к выполнению за последнюю секунду.

Процедура schedcpu() также пересчитывает приоритеты для режима задачи всех процессов по формуле:

p\_usrpri = PUSER + (p\_cpu / 4) + (2 \* p\_nice)

Где PUSER – базовые приоритет в режиме задачи, равный 50. Если процесс в последний раз использовал много процессорного времени, его p\_cpu будет увеличен. Процессы, требующие много операция ввода вывода, имеют высокий приоритет. В то время как процессы, имеющие большое количество операций, имеют более низкий.

### 2.2.1 Фактор полураспада

Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его p\_cpu, что приводит к повышению его приоритета. Такая схема предотвращает зависание низкоприоритетных процессов по вине операционной системы.

Основная идея планирования заключается в хранении приоритетов всех таких процессов примерно в том же диапазоне в течение некоторого периода времени. Приоритеты могут повышаться или понижаться в рамках этого диапазона в зависимости от того, сколько процессорного времени эти процессы получали в последний раз. Фактор полураспада обеспечивает экспоненциально взвешенное среднее значение использования процессора в течение всего периода функционирования процесса.

# Вывод

В операционных системах Windows и Unix/Linux обработчик прерывания от системного таймера выполняет схожие функции такие, как:

* Обновление системного времени
* Уменьшение кванта текущего процесса
* Перепланировка задач
* Отправление отложенных вызовов на выполнение

Однако в планировании и пересчете приоритетов эти системны отличаются. Планировщик Windows вытесняющий, а Unix/Linux строго невытесняющее. Опираясь на эти модели, планирование процессов имеет совершенно разные алгоритмы работы, сохраняя приоритет процесса или потока примерно в одном диапазоне.