# Глава 3. Системные механизмы

Операционная система Windows предоставляет ряд основных механизмов, используемых такими компонентами режима ядра, как исполняющая система, ядро и драйверы устройств. В данной главе рассматриваются следующие системные механизмы и описывается порядок их использования:

- диспетчеризация системных прерываний, включая прерывания, отложенные вызовы процедур (DPC), асинхронные вызовы процедур (APC), диспетчеризация исключений и диспетчеризация системных служб;
- диспетчер объектов исполняющей системы;
- синхронизация, включая спин-блокировки, объекты диспетчера ядра, порядок реализации ожиданий, а также примитивы синхронизации, относящиеся к пользовательскому режиму и избегающие переходов в режим ядра (в отличие от обычных объектов диспетчера);
- системные рабочие потоки;
- различные механизмы, например глобальные флаги Windows;
- усовершенствованные вызовы локальных процедур (ALPC);
- трассировка событий ядра (Kernel event tracing);
- Wow64;
- отладка в пользовательском режиме;
- загрузчик образов (image loader);
- гипервизор (Hyper-V);
- диспетчер транзакций ядра (Kernel Transaction Manager, KTM);
- защита ядра от исправлений (Kernel Patch Protection, KPP);
- обеспечение целостности кода.

## Диспетчеризация системных прерываний

Прерывания и исключения являются условиями операционной системы, отвлекающими процессор на выполнение кода, находящегося за пределами нормального потока управления. Они могут быть обнаружены либо аппаратными, либо программными средствами. Термин *системное прерывание* относится к механизму процессора, предназначенному для захвата выполняемого потока при возникновении исключения или прерывания и для передачи управления в определенное место в операционной системе. В Windows процессор передает управление *обработчику системного прерывания*, который является функцией, характерной для того или иного прерывания или исключения. На рис. 3.1 проиллюстрированы некоторые условия активации обработчиков системных прерываний.

Ядро различает прерывания и исключения следующим образом. *Прерывание* является асинхронным событием (которое может произойти в любое время), не связанным с текущей задачей процессора. Прерывания генерируются, главным образом, устройствами ввода-вывода, процессорными часами или таймерами, и они могут быть разрешены (включены) или запрещены (выключены). *Исключение*, напротив, является синхронным условием, которое обычно возникает в резуль-

тате выполнения конкретной инструкции. (Аварийные завершения работы, например, из-за машинного сбоя, обычно не связаны с выполнением инструкции.) Повторение исключений может быть вызвано повторным запуском программы с теми же данными и при тех же условиях. В качестве примеров исключений можно привести нарушения доступа к памяти, определенные инструкции отладки и ошибки деления на нуль. Ядро также считает исключениями вызовы системных служб (хотя технически они являются системными прерываниями).


**Рис. 3.1.** Диспетчеризация системных прерываний

Исключения и прерывания могут генерироваться как оборудованием, так и программами. Например, причиной исключения, связанного с ошибкой шины, является оборудование, а исключение, связанное с делением на нуль, является результатом программной ошибки. Точно так же прерывание может генерироваться устройством ввода-вывода, или же программное прерывание может быть выдано самим ядром (прерывания APC или DPC будут рассмотрены в этой главе).

При генерации аппаратного исключения или прерывания процессор записывает довольно большой объем информации о состоянии машины в стек ядра того потока, который был прерван, для возвращения к нужной точке потока управления и продолжения выполнения, как будто ничего не случилось. Если поток выполнялся в пользовательском режиме, Windows переключается на стек потоков режима ядра. Затем Windows создает в стеке ядра *фрейм системного прерывания* прерванного потока, в котором она сохраняет состояние выполнения потока. Фрейм системного прерывания является подмножеством полного контекста потока, и его определение можно увидеть, набрав в отладчике ядра команду `dt nt!_ktrap_frame` (см. главу 5). Ядро обрабатывает программные прерывания либо как часть обработки аппаратных прерываний, либо одновременно с ними, когда поток вызывает функции ядра, относящиеся к программному прерыванию.

В большинстве случаев ядро устанавливает внешние функции обработки системных прерываний, которые выполняют общие задачи их обработки до и после передачи управления другим функциям, выставившим системное прерывание. Например, если ситуация была вызвана прерыванием от какого-нибудь устройства, находящийся в ядре обработчик аппаратных системных прерываний передает управление *процедуре обработки прерывания* (Interrupt service routine, ISR), которую драйвер устройства предоставил для устройства, вызвавшего прерывание. Если ситуация создалась из-за вызова системной службы, общий обработчик системных прерываний, связанных с системными службами, передает управление конкретной функции системной службы в исполняющей системе. Ядро также устанавливает обработчики системных прерываний для неожиданных или необрабатываемых им прерываний. Эти обработчики системных прерываний обычно выполняют системную функцию KeBugCheckEx, останавливающую компьютер, когда ядро обнаруживает проблемное или некорректное поведение, которое, если его не предотвратить, может привести к повреждению данных. Более подробно прерывания, исключения и диспетчеризация системных служб рассматриваются в следующих разделах.

## Диспетчеризация прерываний

Аппаратно генерируемые прерывания обычно исходят от устройств ввода-вывода, которые должны уведомить процессор о том, что они нуждаются в обслуживании. Устройства, управляемые прерываниями, позволяют операционной системе использовать процессор максимально эффективно, совмещая основную обработку данных с операциями ввода-вывода. Поток запускает передачу ввода-вывода в адрес устройства или от него, а затем может выполнять другую полезную работу, пока устройство не завершит передачу. Когда устройство завершит работу, оно прерывает процессор на свое обслуживание. Как правило, прерываниями управляются устройства указания координат, принтеры, клавиатуры, дисковые приводы и сетевые карты.

Прерывания могут также генерироваться системными программами. Например, ядро может выдать программное прерывание для инициирования диспетчера потока и для асинхронного проникновения в выполнение потока.

Ядро может также запретить прерывания, но делается это крайне редко, только в критических ситуациях, например во время программирования контроллера прерываний или диспетчеризации исключений. Для ответа на прерывания, исходящие от устройств, ядро устанавливает обработчики системных прерываний. Эти обработчики передают управление либо внешней процедуре (ISR), обрабатывающей прерывание, либо внутренней процедуре ядра, реагирующей на прерывание. Для обслуживания прерываний от устройств драйверы устройств предоставляют ISR-процедуры, а ядро предоставляет процедуры, обрабатывающие другие типы прерываний.

В следующих разделах вы узнаете, как оборудование уведомляет процессор о прерываниях, исходящих от устройств, о типах прерываний, поддерживаемых ядром, о том, как драйверы устройств взаимодействуют с ядром (в том, что касается обработки прерываний), о том, какие программные прерывания распознаются ядром, а также об объектах ядра, используемых для их реализации.

## Обработка аппаратных прерываний

На аппаратных платформах, поддерживаемых Windows, внешние прерывания ввода-вывода поступают на одну из линий контроллера прерываний. В свою очередь, контроллер прерывает работу процессора по отдельной линии. Как только работа процессора будет прервана, этот процессор требует от контроллера запрос прерывания (Interrupt request, IRQ). Контроллер прерываний превращает IRQ в номер прерывания, использует этот номер в качестве индекса в структуре под названием *таблица диспетчеризации прерываний* (Interrupt dispatch table, IDT) и передает управление соответствующей процедуре обработки прерывания. Во время загрузки системы Windows заполняет IDT указателями на процедуры ядра, обрабатывающими каждое прерывание и исключение.

Windows отображает аппаратные IRQ-запросы на номера прерываний в IDT и также использует IDT для настройки обработчиков системных прерываний для исключений. Например, в системах x86 и x64 номер исключения для ошибки отсутствия страницы (исключения, которое возникает, когда поток пытается обратиться к странице виртуальной памяти, которая не определена или отсутствует) имеет значение 0xe (14). Таким образом, запись 0xe в IDT указывает на системный обработчик ошибки обращения к странице. Хотя архитектура, поддерживаемая Windows, допускает до 256 записей в IDT-таблице, количество IRQ-запросов, поддерживаемых на конкретной машине, определяется конструкцией используемого контроллера прерываний.

### ЭКСПЕРИМЕНТ: ПРОСМОТР IDT

Содержимое IDT, включая информацию о том, какие обработчики системных прерываний назначены операционной системой Windows прерываниям (включая исключения и IRQ-запросы), можно посмотреть, используя команду отладки ядра !idt. Команда !idt без ключей показывает упрощенный вывод, который включает только зарегистрированные аппаратные прерывания (и на 64-разрядных машинах обработчики системных прерываний процессора).

В следующем примере показано, как выглядит вывод команды !idt:

```
lkd> !idt
Dumping IDT:
00:  fffff80001a7ec40 nt!KiDivideErrorFault
01:  fffff80001a7ed40 nt!KiDebugTrapOrFault
02:  fffff80001a7ef00 nt!KiNmiInterrupt  Stack = 0xFFFFF80001865000
03:  fffff80001a7f280 nt!KiBreakpointTrap
04:  fffff80001a7f380 nt!KiOverflowTrap
05:  fffff80001a7f480 nt!KiBoundFault
06:  fffff80001a7f580 nt!KiInvalidOpcodeFault
07:  fffff80001a7f7c0 nt!KiNpxNotAvailableFault
08:  fffff80001a7f880 nt!KiDoubleFaultAbort  Stack = 0xFFFFF80001863000
09:  fffff80001a7f940 nt!KiNpxSegmentOverrunAbort
0a:  fffff80001a7fa00 nt!KiInvalidTssFault
0b:  fffff80001a7fac0 nt!KiSegmentNotPresentFault
0c:  fffff80001a7fc00 nt!KiStackFault
0d:  fffff80001a7fd40 nt!KiGeneralProtectionFault
```

*продолжение* ⇩

```
0e:  fffff80001a7fe80 nt!KiPageFault
10:  fffff80001a80240 nt!KiFloatingErrorFault
11:  fffff80001a803c0 nt!KiAlignmentFault
12:  fffff80001a804c0 nt!KiMcheckAbort  Stack = 0xFFFFF80001867000
13:  fffff80001a80840 nt!KiXmmException
1f:  fffff80001a5ec10 nt!KiApcInterrupt
2c:  fffff80001a80a00 nt!KiRaiseAssertion
2d:  fffff80001a80b00 nt!KiDebugServiceTrap
2f:  fffff80001acd590 nt!KiDpcInterrupt
37:  fffff8000201c090 hal!PicSpuriousService37 (KINTERRUPT fffff8000201c000)
3f:  fffff8000201c130 hal!PicSpuriousService37 (KINTERRUPT fffff8000201c0a0)
51:  fffffa80045babd0 dxgkrnl!DpiFdoLineInterruptRoutine (KINTERRUPT fffffa80045bab40)
52:  fffffa80029f1390 USBPORT!USBPORT_InterruptService (KINTERRUPT fffffa80029f1300)
62:  fffffa80029f15d0 USBPORT!USBPORT_InterruptService (KINTERRUPT fffffa80029f1540)
           USBPORT!USBPORT_InterruptService (KINTERRUPT fffffa80029f1240)
72:  fffffa80029f1e10 ataport!IdePortInterrupt (KINTERRUPT fffffa80029f1d80)
81:  fffffa80045bae10 i8042prt!I8042KeyboardInterruptService (KINTERRUPT
     fffffa80045bad80)
82:  fffffa80029f1ed0 ataport!IdePortInterrupt (KINTERRUPT fffffa80029f1e40)
90:  fffffa80045bad50 Vid+0x7918 (KINTERRUPT fffffa80045bacc0)
91:  fffffa80045baed0 i8042prt!I8042MouseInterruptService (KINTERRUPT fffffa80045bae40)
a0:  fffffa80045bac90 vmbus!XPartPncIsr (KINTERRUPT fffffa80045bac00)
a2:  fffffa80029f1210 sdbus!SdbusInterrupt (KINTERRUPT fffffa80029f1180)
           rimmpx64+0x9FFC (KINTERRUPT fffffa80029f10c0)
           rimspx64+0x7A14 (KINTERRUPT fffffa80029f1000)
           rixdpx64+0x9C50 (KINTERRUPT fffffa80045baf00)
a3:  fffffa80029f1510 USBPORT!USBPORT_InterruptService (KINTERRUPT fffffa80029f1480)
           HDAudBus!HdaController::Isr (KINTERRUPT fffffa80029f1c00)
a8:  fffffa80029f1bd0 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f1b40)
a9:  fffffa80029f1b10 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f1a80)
aa:  fffffa80029f1a50 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f19c0)
ab:  fffffa80029f1990 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f1900)
ac:  fffffa80029f18d0 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f1840)
ad:  fffffa80029f1810 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f1780)
ae:  fffffa80029f1750 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f16c0)
af:  fffffa80029f1690 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f1600)
b0:  fffffa80029f1d50 NDIS!ndisMiniportMessageIsr (KINTERRUPT fffffa80029f1cc0)
b1:  fffffa80029f1f90 ACPI!ACPIInterruptServiceRoutine (KINTERRUPT fffffa80029f1f00)
b3:  fffffa80029f1450 USBPORT!USBPORT_InterruptService (KINTERRUPT fffffa80029f13c0)
c1:  fffff8000201c3b0 hal!HalpBroadcastCallService (KINTERRUPT fffff8000201c320)
d1:  fffff8000201c450 hal!HalpHpetClockInterrupt (KINTERRUPT fffff8000201c3c0)
d2:  fffff8000201c4f0 hal!HalpHpetRolloverInterrupt (KINTERRUPT fffff8000201c460)
df:  fffff8000201c310 hal!HalpApicRebootService (KINTERRUPT fffff8000201c280)
e1:  fffff80001a8e1f0 nt!KiIpiInterrupt
e2:  fffff8000201c270 hal!HalpDeferredRecoveryService (KINTERRUPT fffff8000201c1e0)
e3:  fffff8000201c1d0 hal!HalpLocalApicErrorService (KINTERRUPT fffff8000201c140)
fd:  fffff8000201c590 hal!HalpProfileInterrupt (KINTERRUPT fffff8000201c500)
fe:  fffff8000201c630 hal!HalpPerfInterrupt (KINTERRUPT fffff8000201c5a0)
```

На системе, предоставившей вывод для данного эксперимента, ISR-процедура клавиатуры в драйвере устройства (I8042prt.sys) назначена прерыванию с номером 0x81. Можно также увидеть, что как ранее было объяснено, прерывание 0xe соответствует системной функции KiPageFault. ■

Каждый процессор имеет отдельную IDT-таблицу, поэтому другие процессоры, если нужно, могут запускать другие ISR-процедуры. Например, в мультипроцессорной системе прерывание от часов получает каждый процессор, но только один процессор в ответ на это прерывание обновляет показания системных часов. Тем не менее все процессоры используют прерывание для замера кванта времени потока и для инициации перепланирования по истечении этого кванта.

Кроме того, некоторые настройки системы могут требовать обработку определенных прерываний от устройств конкретным процессором.

## Контроллер прерываний x86

В большинстве систем x86 используется либо программируемый контроллер прерываний (Programmable Interrupt Controller, PIC) i8259A, либо один из вариантов усовершенствованного программируемого контроллера прерываний (Advanced Programmable Interrupt Controller, APIC) i82489. Современные контроллеры комплектуются контроллером APIC. Стандарт PIC возник с появлением IBM PC. PIC i8259A работает только с однопроцессорными системами и имеет только 8 линий прерываний. Но в архитектуре IBM PC определена возможность использования дополнительного второго PIC-контроллера, называемого ведомым, чьи прерывания мультиплексируются в одну из линий прерываний ведущего PIC-контроллера. Тем самым общее число прерываний доводится до 15 (7 на ведущем и 8 на ведомом контроллере, мультиплексируемые на восьмую линию прерывания ведущего контроллера). С мультипроцессорными системами работают APIC- и SAPIC-контроллеры (Streamlined Advanced Programmable Interrupt Controllers — модернизированные усовершенствованные программируемые контроллеры прерываний — мы вскоре рассмотрим), имеющие 256 линий прерываний. Intel и ряд других компаний определили мультипроцессорную спецификацию — Multiprocessor Specification (MP Specification), конструкторский стандарт для мультипроцессорных систем x86, в основу которого заложено применение APIC. Для обеспечения совместимости с однопроцессорными операционными системами и кодом начальной загрузки, который запускает мультипроцессорную систему в однопроцессорном режиме, APIC-контроллеры поддерживают режим совместимости с 15 прерываниями и доставки прерываний только основному процессору. Архитектура APIC-контроллера изображена на рис. 3.2.

Фактически APIC-контроллер состоит из нескольких компонентов: APIC ввода-вывода, получающего прерывания от устройств, локальных APIC-контроллеров, получающих прерывания от APIC ввода-вывода на шине и прерывающих работу того центрального процессора, с которым они связаны, и i8259A-совместимого контроллера прерываний, который переводит APIC-ввод в сигналы, эквивалентные PIC-контроллеру. Поскольку в системе может быть несколько APIC-контроллеров ввода-вывода, у материнских плат обычно есть часть основных логических устройств, расположенных между ними и процессорами. Эти логические устройства отвечают за реализацию алгоритмов процедур прерываний, которые наряду с балансировкой нагрузки аппаратных прерываний между

процессорами и попыткой получить преимущества от локализации, доставляют прерывания от устройств тому же самому процессору, которому только что было направлено предыдущее прерывание того же типа. Обычные программы могут перепрограммировать APIC-контроллеры ввода-вывода, применив фиксированный алгоритм маршрутизации, обходящий логику набора микросхем. Windows делает это путем программирования APIC-контроллеров в режиме маршрутизации «прерывания одного процессора в следующем наборе».

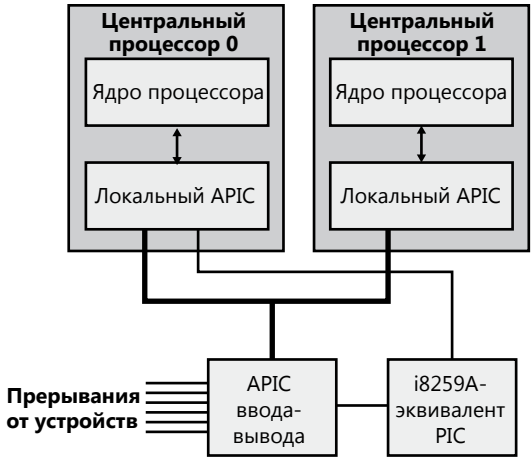


**Рис. 3.2.** APIC-архитектура x86

## Контроллеры прерываний x64

Поскольку архитектура x64 совместима с операционными системами x86, системы x64 должны предоставлять такие же контроллеры прерываний, что и системы x86. Но существенная разница состоит в том, что x64-версии не будут запускаться на системах, которые не имеют APIC, поскольку для управления прерываниями они используют APIC.

## Контроллеры прерываний IA64

Архитектура IA64 зависит от модернизированного усовершенствованного программируемого контроллера Streamlined Advanced Programmable Interrupt Controller (SAPIC), который является результатом развития контроллера APIC. Даже если во встроенной программе присутствует балансировка и маршрутизация, Windows ими не пользуется, вместо этого она назначает прерывания процессорам статически, по кругу.

### ЭКСПЕРИМЕНТ: ПРОСМОТР PIC И APIC

Конфигурацию PIC на однопроцессорной системе и текущего локального APIC на мультипроцессорной системе можно просмотреть соответственно с помощью команд отладчика ядра !pic и !apic. Вывод команды !pic на однопроцессорной системе имеет следующий вид[1].

[1] Следует учесть, что команда !pic в системе, использующей APIC HAL, не работает.

```
lkd> !pic
----- IRQ Number ----- 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Physically in service:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
Physically masked:      .  .  .  Y  .  .  Y  Y  .  .  Y  .  .  Y  .  .
Physically requested:   .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
Level Triggered:        .  .  .  .  .  Y  .  .  .  Y  .  Y  .  .  .  .
```

Далее следует вывод команды !apic, запущенной на системе, использующей APIC HAL. Учтите, что при локальной отладке ядра эта команда показывает APIC, связанный с текущим процессором, — иначе говоря, с тем процессором, на котором был запущен поток отладчика при вводе команды. При просмотре аварийного дампа или удаленной системы для переключения на тот процессор, чей локальный APIC нужно просмотреть, можно использовать команду ~ (тильда) с указанием после нее номера процессора.

```
lkd> !apic
Apic @ fffe0000  ID:0 (50014)  LogDesc:01000000  DestFmt:ffffffff  TPR 20
TimeCnt: 00000000clk  SpurVec:3f  FaultVec:e3  error:0
Ipi Cmd: 01000000'0000002f  Vec:2F  FixedDel    Ph:01000000      edg high
Timer..: 00000000'000300fd  Vec:FD  FixedDel    Dest=Self      edg high      m
Linti0.: 00000000'0001003f  Vec:3F  FixedDel    Dest=Self      edg high      m
Linti1.: 00000000'000004ff  Vec:FF  NMI         Dest=Self      edg high
TMR: 51-52, 62, A3, B1, B3
IRR:
ISR::
```

Различные числа, следующие за метками Vec, показывают вектор в IDT, связанный с заданной командой. Например, в данном выводе прерывание номер 0xFD связано с APIC Timer, а прерывание номер 0xE3 управляет ошибками APIC. Поскольку этот эксперимент был запущен на той же самой машине, что и ранее проводившийся эксперимент !idt, можно заметить, что 0xFD является прерыванием профилирования HAL (которое использует таймер для профилирования интервалов), а 0xe3, как и ожидалось, является обработчиком ошибок локального APIC HAL.

Следующий вывод является результатом выполнения команды !ioapic, показывающей конфигурацию APIC-контроллеров ввода-вывода, компонентов контроллера прерываний, подключенного к устройствам:

```
lkd> !ioapic
IoApic @ FEC00000  ID:0 (51)  Arb:A951

Inti00.: 0000a951'0000a951  Vec:51  LowestDl  Lg:0000a951      lvl low     ■
```

## Уровни запросов программных прерываний (IRQL)

Хотя контроллеры прерываний устанавливают приоритетность прерываний, Windows устанавливает свою собственную схему приоритетности прерываний, известную как *уровни запросов прерываний* (IRQL). В ядре IRQL-уровни представлены в виде номеров от 0 до 31 на системах x86 и в виде номеров от 0 до 15 на системах x64 и IA64, где более высоким номерам соответствуют прерывания с более высоким приоритетом. Хотя ядро определяет для программных прерываний стандартный набор IRQL-уровней, HAL отображает номера аппаратных пре-

рываний на IRQL-уровни. На рис. 3.3 показаны IRQL-уровни для архитектуры x86, а на рис. 3.4 показаны IRQL-уровни для архитектур x64 и IA64.

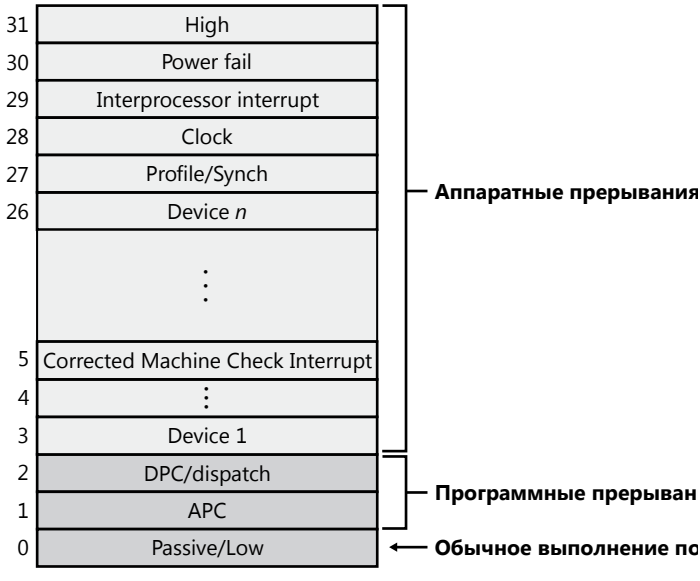


**Рис. 3.3.** Уровни запросов прерываний (IRQL) для архитектуры x86

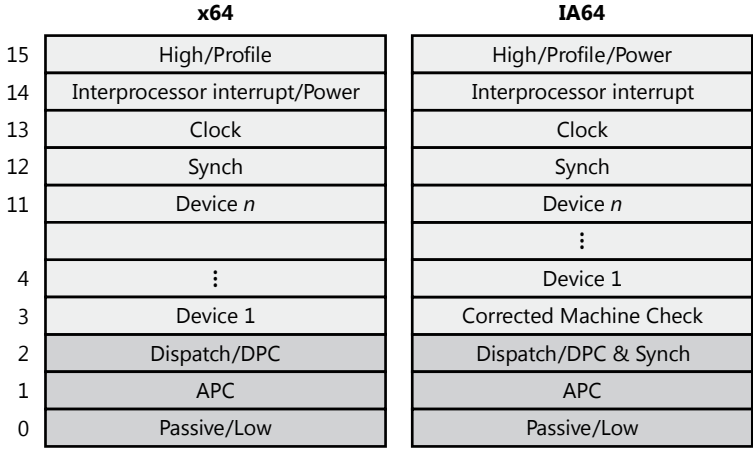


**Рис. 3.4.** Уровни запросов прерываний (IRQL) для архитектур x64 и IA64

Прерывания обслуживаются в порядке их приоритета, и прерывания с более высоким уровнем приоритета получают преимущество в обслуживании. При возникновении прерывания с высоким уровнем процессор сохраняет состояние прерванного потока и запускает связанный с прерыванием диспетчер системных прерываний. Тот, в свою очередь, поднимает IRQL и вызывает процедуру обработки прерывания. После выполнения этой процедуры диспетчер прерываний понижает IRQL-уровень процессора до значения, на котором он был до

возникновения прерывания, а затем загружает сохраненное состояние машины. Прерванный поток продолжает выполнение с того места, в котором оно было прервано. Когда ядро понижает IRQL, могут реализоваться те прерывания с более низким уровнем приоритета, которые были замаскированы. Если так и происходит, ядро повторяет процесс для обработки новых прерываний.

Уровни приоритетов IRQL имеют совершенно другое значение, чем приоритеты, используемые при планировании потоков (которые рассматриваются в главе 5). Приоритет планирования является атрибутом потока, а IRQL является атрибутом источника прерывания, такого как клавиатура или мышь. Кроме того, у каждого процессора есть установка IRQL, которая меняется при выполнении кода операционной системы.

Установка IRQL каждого процессора определяет, какие прерывания данный процессор может получать. IRQL-уровни также используются для синхронизации доступа к структуре данных режима ядра. Как только запускается поток режима ядра, он повышает или понижает IRQL процессора либо напрямую, путем вызова функций KeRaiseIrql и KeLowerIrql, либо, что случается чаще, опосредованно, через вызовы функций, которые запрашивают объекты ядра, используемые для синхронизации. Как показано на рис. 3.5, прерывания, поступающие от источника с IRQL, превышающим текущий уровень, прерывают работу процессора, а прерывания, поступающие от источников с IRQL-уровнями равными или ниже текущего уровня, *маскируются* до тех пор, пока выполняющийся поток не понизит IRQL.

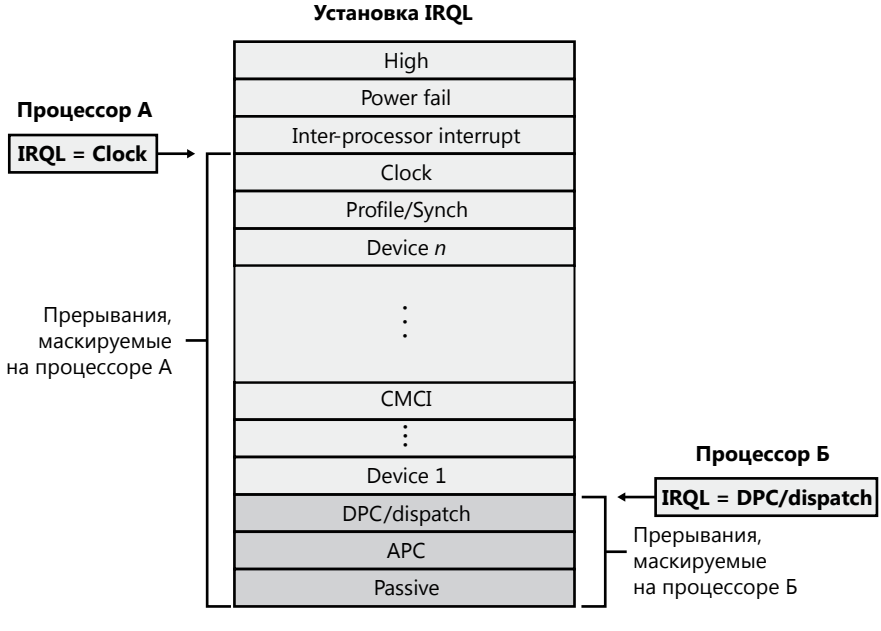


**Рис. 3.5.** Маскирование прерываний

Поскольку обращение к PIC является относительно медленной операцией, HAL-механизмы, требующие доступа к шине ввода-вывода для изменения IRQL-уровней (например, для PIC и 32-разрядных систем усовершенствованного интерфейса управления конфигурированием и энергопотреблением — Advanced Configuration

and Power Interface, ACPI), реализуют оптимизацию производительности, которая называется «ленивой IRQL» (lazy IRQL) и позволяет избежать обращений к PIC. При повышении IRQL HAL отмечает для себя новый IRQL, не изменяя маски прерывания. Если же после этого произойдет прерывание с более низким уровнем, HAL устанавливает маску прерывания с настройками, соответствующими первому прерыванию, и не замораживает прерывание с более низким уровнем (сохраняя его тем самым в отложенном состоянии) до тех пор, пока IRQL не будет понижен. Таким образом, если прерываний с более низким уровнем при повышении IRQL не происходит, модифицировать PIC HAL-механизмам не требуется.

### ПРИМЕЧАНИЕ

Исключения из правила блокировки прерываний равного или более низкого уровня при повышении IRQL касаются прерываний APC-уровня. Если поток повышает IRQL до уровня APC, а затем его выполнение подвергается перепланированию из-за прерывания dispatch/DPC-уровня, система может передать прерывание APC-уровня заново спланированному потоку. Таким образом, уровень APC может считаться IRQL, локальным для потока, а не для всего процессора.

Поток режима ядра повышает или понижает IRQL того процессора, на котором он запущен, в зависимости от того, что он пытается сделать. Например, когда возникает прерывание, обработчик системных прерываний (или, возможно, процессор) повышает IRQL процессора до IRQL, установленного для источника прерывания. Это повышение маскирует все прерывания с таким же и более низким IRQL (только на этом процессоре), гарантируя, что процессор, обслуживающий прерывание, не захватывается прерываниями на том же или на более низком уровне. Замаскированные прерывания либо обрабатываются другим процессором, либо придерживаются до тех пор, пока IRQL не упадет. Поэтому все компоненты системы, включая ядро и драйверы устройств, стараются держать IRQL на пассивном уровне (который иногда называют низким уровнем). Это сделано для своевременной реакции драйверов устройств на аппаратные прерывания при условии, что IRQL не держится неоправданно высоким в течение продолжительных периодов времени.

### ЭКСПЕРИМЕНТ: ПРОСМОТР IRQL

Просмотреть сохраненный для процессора IRQL можно с помощью команды отладчика !irql. Сохраненный IRQL представляет собой IRQL на момент, непосредственно предшествующий проникновению в отладчик, повышающему IRQL до статического, ничего не значащего значения:

```
kd> !irql
Debugger saved IRQL for processor 0x0 -- 0 (LOW_LEVEL)
```

Учтите, что значение IRQL сохраняется в двух местах. Первое место, в котором представлен текущий IRQL, — это область управления процессором (processor control region, PCR), а второе — его расширение, блок управления областью процессора (processor region control block, PRCB), который содержит сохраненный IRQL в поле DebuggerSaveIrql. PCR и PRCB содержат информацию о состоянии каждого процессора в системе, в том числе текущий IRQL, указатель на аппаратную IDT-таблицу, сведения о текущем потоке

и следующем, выбранном для запуска потоке. Ядро и HAL используют эту информацию для выполнения действий, ориентированных на конкретную архитектуру и конкретную машину. Части структур PCR и PRCB открыто определены в заголовочном файле Ntddk.h, относящемся к инструментарию Windows Driver Kit (WDK).

Содержимое PCR текущего процессора можно просмотреть с помощью отладчика ядра, используя команду !pcr. Для просмотра PCR конкретного процессора нужно после команды добавить номер процессора, отделив его от команды пробелом:

```
lkd> !pcr 0
KPCR for Processor 0 at  fffff80001bfad00:
    Major 1 Minor 1
    NtTib.ExceptionList: fffff80001853000
        NtTib.StackBase: fffff80001854080
       NtTib.StackLimit: 000000000026ea28
     NtTib.SubSystemTib: fffff80001bfad00
          NtTib.Version: 0000000001bfae80
      NtTib.UserPointer: fffff80001bfb4f0
          NtTib.SelfTib: 000007fffffdb000
                SelfPcr: 0000000000000000
                   Prcb: fffff80001bfae80
                   Irql: 0000000000000000
                    IRR: 0000000000000000
                    IDR: 0000000000000000
          InterruptMode: 0000000000000000
                    IDT: 0000000000000000
                    GDT: 0000000000000000
                    TSS: 0000000000000000
          CurrentThread: fffff80001c08c40
             NextThread: 0000000000000000
             IdleThread: fffff80001c08c40
              DpcQueue:
```

■

Поскольку изменение IRQL процессора существенно влияет на работу системы, это изменение должно вноситься только в режиме ядра — потоки пользовательского режима внести это изменение не могут. Это означает, что IRQL процессора при выполнении кода в пользовательском режиме всегда находится на пассивном уровне. Уровень IRQL может быть повышен только при выполнении кода в режиме ядра.

У каждого уровня прерывания есть своя конкретная цель. Например, ядро выдает *межпроцессорное прерывание* (Interprocessor interrupt, IPI), чтобы запросить выполнение действия на другом процессоре, например диспетчеризацию конкретного потока для выполнения или обновления кэш-памяти его буфера быстрого преобразования адреса (Translation look-aside buffer, TLB). Через определенные периоды времени системные часы генерируют прерывание, а ядро реагирует на него, обновляя значение часов и измеряя время выполнения потока. Если аппаратная платформа поддерживает двое часов, ядро добавляет еще один уровень прерывания от системных часов для измерения производительности.