**Прерывания от внешних устройств в системе x86. Часть 1. Эволюция контроллеров прерываний**

В данной статье хотелось бы рассмотреть механизмы доставки прерываний от внешних устройств в системе x86 и попытаться ответить на вопросы:

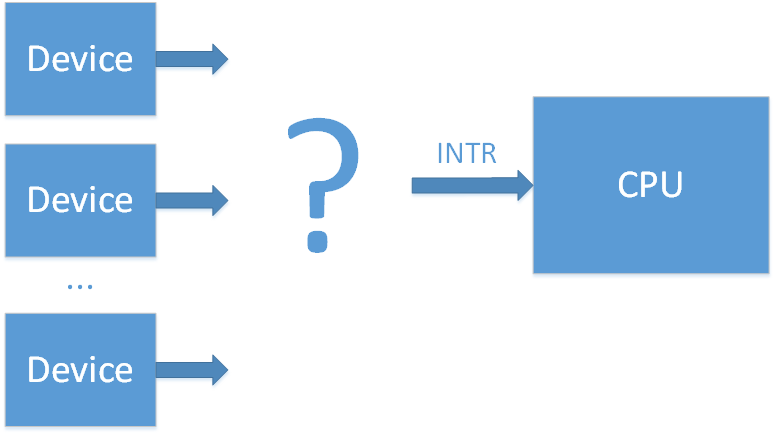
* что такое PIC и для чего он нужен?
* что такое APIC и для чего он нужен? Для чего нужны LAPIC и I/O APIC?
* в чём отличия APIC, xAPIC и x2APIC?
* что такое MSI? В чём отличия MSI и MSI-X?
* как с этим связаны таблицы $PIR, MPtable, ACPI?  
  Если на какой-то из этих вопросов вам интересно получить ответ или вы просто хотите ознакомиться с эволюцией контроллеров прерываний в системе x86, добро пожаловать под кат.

Введение

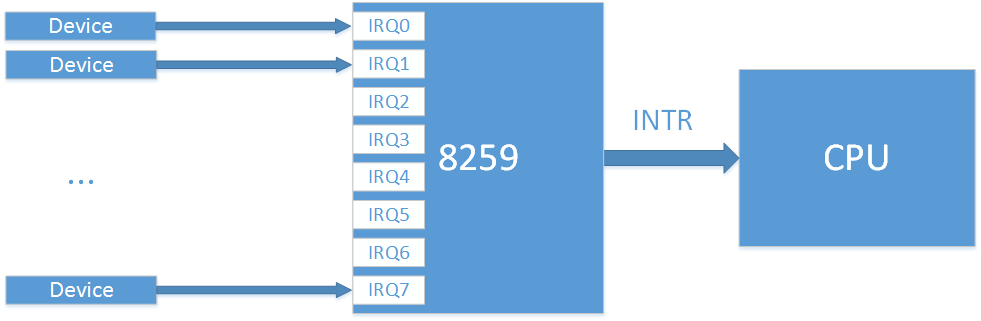
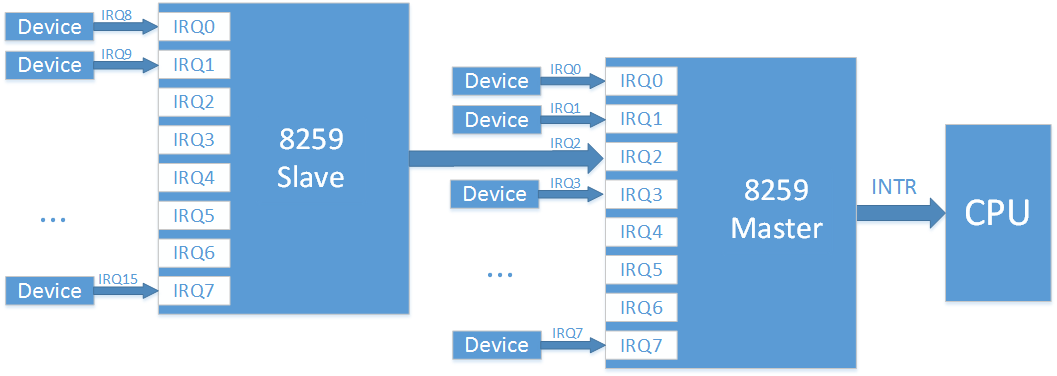
Все мы знаем, что такое прерывание. Для тех, кто нет, цитата из википедии:

Прерывание (англ. interrupt) — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерывания (или программой обработки прерывания), которая реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.  
  
В зависимости от источника возникновения сигнала прерывания делятся на:

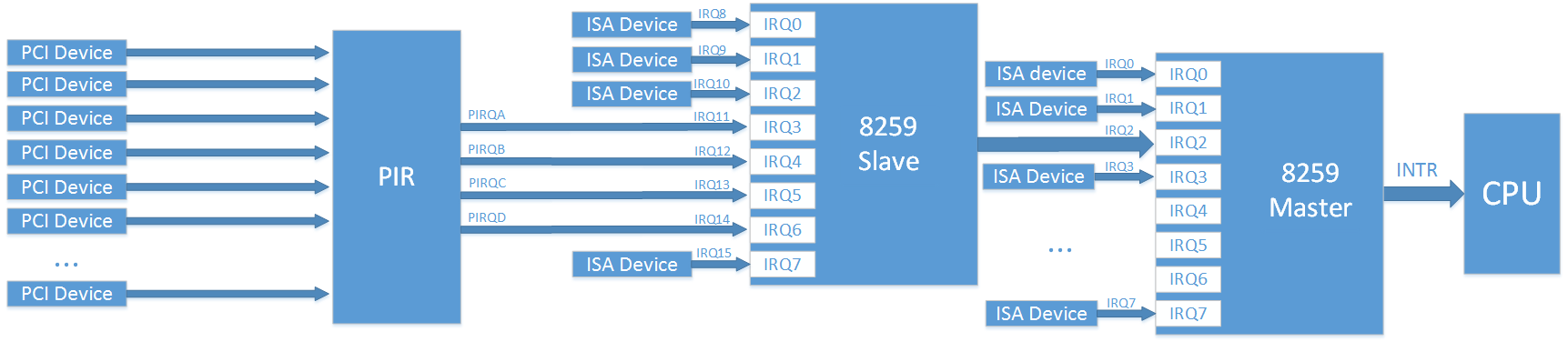
* асинхронные, или внешние (аппаратные) — события, которые исходят от внешних аппаратных устройств (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, сетевой карты или дискового накопителя, нажатие клавиш клавиатуры, движение мыши. Факт возникновения в системе такого прерывания трактуется как запрос на прерывание (англ. Interrupt request, IRQ) — устройства сообщают, что они требуют внимания со стороны ОС;
* синхронные, или внутренние — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода: деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции;

В данной статье хотелось бы обсудить внешние прерывания IRQ.  
  
Зачем они нужны? Допустим мы хотим выполнить какое-либо действие со входным пакетом для сетевой карты, когда он придёт. Чтобы не спрашивать сетевую карту постоянно «есть ли у тебя новый пакет?» и не тратить на это ресурсы процессора, можно использовать прерывание IRQ. Линия прерываний устройства соединяется с линией INTR процессора, и при получении пакета сетевая карта «дергает» эту линию. Процессор понимает, что для него есть информация и читает пакет.  
  
Но что делать если устройств много? На все внешние устройства ножек процессора не напасёшься.   
  
  
  
Чтобы решить эту проблему, придумали микросхему — контроллер прерываний.

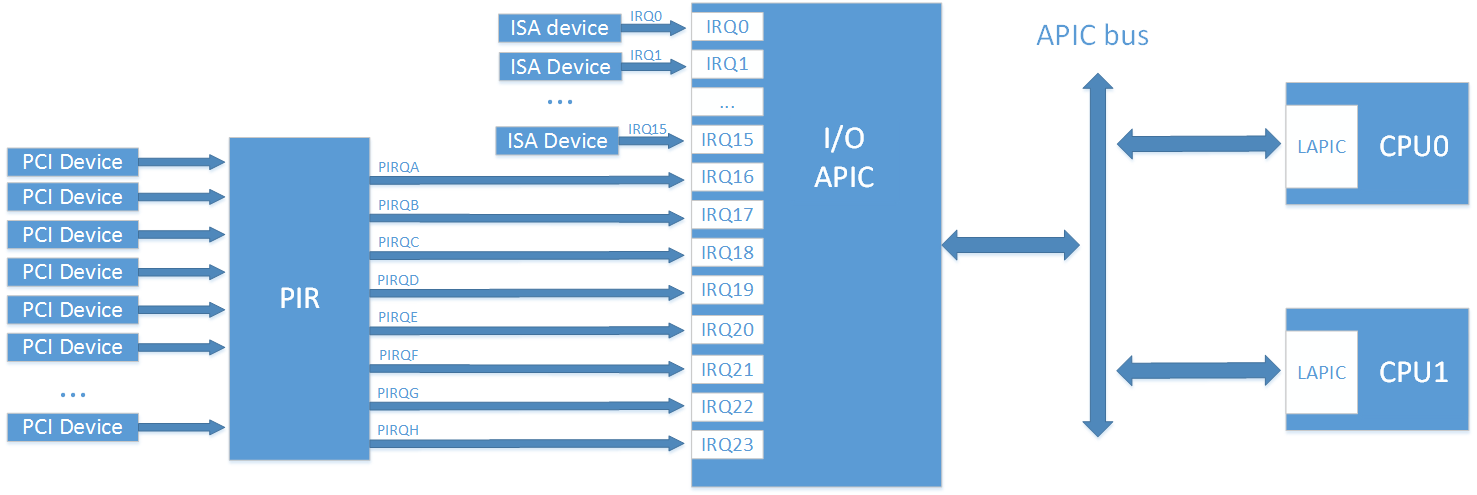
PIC

([вики](https://en.wikipedia.org/wiki/Programmable_interrupt_controller)/[osdev](https://wiki.osdev.org/8259_PIC))   
  
Первой была микросхема [Intel 8259 PIC](https://en.wikipedia.org/wiki/Intel_8259). 8 входных линий (IRQ0-7), и одна выходная, соединяющая контроллер с линией INTR процессора. Когда возникает прерывание от какого-либо устройства, 8259 дёргает линию INTR, процессор понимает, что какое-то устройство сигнализирует о прерывании и опрашивает PIC, чтобы понять по какой именно ножке IRQx возникло прерывание. Появляется дополнительная задержка на данный опрос, но зато количество линий прерываний увеличивается до 8.  
  
  
  
Однако 8 линий быстро оказалось мало, и чтобы увеличить их количество стали использовать 2 контроллера 8259 (master и slave) соединённых каскадно (Dual PIC).   
  
IRQ с 0 по 7 обрабатываются первым Intel 8259 PIC (master), а IRQ с 8 по 15 вторым 8259 PIC (slave). О возникновении прерывания CPU сигнализирует только master. Если возникло прерывание на линиях 8-15, второй PIC (slave) сигнализирует о прерывании мастеру по линии IRQ 2, и тот уже в свою очередь сигнализирует CPU. Это каскадное прерывание отнимает одну из 16 линий, но в итоге даёт 15 доступных прерываний для устройств.  
  
  
  
Схема утвердилась, и именно её имеют ввиду, когда говорят сейчас о PIC (Programm Interrupt Controller). Впоследствии контроллеры 8259 получили некоторые улучшения, и стали называться 8259A, а эта схема вошла в состав чипсета. Во времена когда основной шиной для подключения внешних устройств была шина ISA, такой системы в целом хватало. Надо было лишь следить, чтобы разные устройства не подключались на одну линию IRQ для избежания конфликтов, так как прерывания ISA не разделяемые.  
  
Обычно раскладка прерываний под устройства была более менее стандартная   
  
Пример (взят [отсюда](https://en.wikipedia.org/wiki/Interrupt_request_(PC_architecture))):  
IRQ 0 — system timer  
IRQ 1 — keyboard controller  
IRQ 2 — cascade (прерывание от slave контроллера)  
IRQ 3 — serial port COM2  
IRQ 4 — serial port COM1  
IRQ 5 — parallel port 2 and 3 or sound card  
IRQ 6 — floppy controller  
IRQ 7 — parallel port 1  
IRQ 8 — RTC timer  
IRQ 9 — ACPI  
IRQ 10 — open/SCSI/NIC  
IRQ 11 — open/SCSI/NIC  
IRQ 12 — mouse controller  
IRQ 13 — math co-processor  
IRQ 14 — ATA channel 1  
IRQ 15 — ATA channel 2  
  
Конфигурация и работа с микросхемами 8259 осуществляется через I/O порты:

|  |  |  |
| --- | --- | --- |
| **Чип** | **Регистр** | **I/O port** |
| Master PIC | Command | 0x0020 |
| Master PIC | Data | 0x0021 |
| Slave PIC | Command | 0x00A0 |
| Slave PIC | Data | 0x00A1 |

→Документацию на 8259A можно найти [тут](https://pdos.csail.mit.edu/6.828/2005/readings/hardware/8259A.pdf)  
  
На смену шине ISA пришла шина PCI. И количество устройств явно стало превосходить число 15, плюс в отличие от статической шины ISA в данном случае случае устройства могут добавляться в систему динамически. Но к счастью в данной шине прерывания могут быть разделяемыми (то есть к одной линии IRQ можно подсоединить несколько устройств). В итоге чтобы решить проблему нехватки линий IRQ, прерывания ото всех PCI устройств решили группировать в линии PIRQ (Programmable Interrupt Request).  
  
Допустим у нас 4 линии прерываний свободно на PIC контроллере, а PCI устройств 20 штук. Мы объединяем прерывания по 5 устройств на линию PIRQx и подключаем линии PIRQx к контроллеру. При возникновении прерывания на линии PIRQx процессору придётся опросить все устройства подключённые к данной линии, чтобы понять от кого именно пришло прерывание, но в целом это решает задачу. Устройство осуществляющее связывание линий прерываний PCI в линии PIRQ часто называют PIR router.   
  
В данном методе надо следить, чтобы линии PIRQx не подсоединялись к линиям IRQx на которых уже заведены прерывания ISA (так как это вызовет конфликты), и чтобы линии PIRQx были сбалансированы (ведь чем больше устройств мы подключили к одной линии PIRQ, тем больше устройств надо будет опрашивать процессору, чтобы понять, какое именно из этих устройств вызвало прерывание).  
  
  
  
**Замечание**: на рисунке маппинг PCI device -> PIR изображён абстрактно, потому что на самом деле он несколько сложнее. В реальности каждый PCI device имеет 4 линии прерываний (INTA, INTB, INTC, INTD). У каждого PCI устройства (device) может быть до 8 функций (functions) и вот каждой функции соответствует уже одно прерывание INTx. Какую именно INTx будет дёргать каждая функция устройства определяется конфигурацией чипсета.  
  
По сути функции это отдельные логические блоки. Например в одном PCI устройстве может быть функция Smbus controller, функция SATA controller, функция LPC bridge. Со стороны ОС каждая функция — это как отдельное устройство со своим конфигурационным пространством PCI Config.   
  
Информацию о роутинге прерываний на PIC контроллере BIOS передавал ОС с помощью таблицы $PIR и с помощью заполнения регистров 3Ch (INT\_LN Interrupt Line (R/W)) и 3Dh (INT\_PN Interrupt Pin (RO)) конфигурационного пространства PCI для каждой функции. Спецификация о таблице $PIR раньше была [на сайте Microsoft](http://www.microsoft.com/whdc/archive/pciirq.mspx), но сейчас её там уже нет. Содержимое строк таблицы $PIR можно понять из [PCI BIOS Specification](http://www.o3one.org/hwdocs/bios_doc/pci_bios_21.pdf) [4.2.2. Get PCI Interrupt Routing Options] или почитать вот [тут](http://ru.osdev.wikia.com/wiki/PCI_IRQ_Routing_Table_Specification)

APIC

([вики](https://en.wikipedia.org/wiki/Advanced_Programmable_Interrupt_Controller), [osdev](https://wiki.osdev.org/APIC))  
  
Предыдущий метод работал пока не появились многопроцессорные системы. Дело в том, что по своему устройству PIC может передавать прерывания только на один главный процессор. А хотелось бы, чтобы нагрузка на процессоры от обработки прерываний была сбалансированной. Решением данной задачи стал новый интерфейс APIC (Advanced PIC).  
  
Для каждого процессора добавляется специальный контроллер LAPIC (Local APIC) и для маршрутизации прерываний от устройств добавляется контроллер [I/O APIC](https://wiki.osdev.org/IOAPIC). Все эти контроллеры объединяются в общую шину с названием APIC (новые системы сейчас уже соединяются по стандартной системной шине).   
  
Когда прерывание от устройства приходит на вывод I/O APIC, контроллер направляет прерывание в LAPIC одного из процессоров. Наличие I/O APIC позволяет сбалансировано распределять прерывания от внешних устройств между процессорами.  
  
Первой микросхемой APIC был [82489DX](https://4donline.ihs.com/images/VipMasterIC/IC/INTL/INTLD047/INTLD047-2-1259.pdf?hkey=EF798316E3902B6ED9A73243A3159BB0), это был отдельный чип, соединяющий в себе LAPIC и I/O APIC. Для создания системы из 2 процессоров нужно было 3 таких микросхемы. 2 функционировали бы как LAPIC и одна как I/O APIC. Позднее функциональность LAPIC была напрямую включена в процессоры, а функциональность I/O APIC была оформлена в чип 82093AA.  
  
I/O APIC [82093AA](https://pdos.csail.mit.edu/6.828/2016/readings/ia32/ioapic.pdf) содержала 24 входных вывода, а архитектура APIC могла поддерживать до 16 CPU. Для поддержки совместимости со старыми системами, прерывания 0~15 отвели под старые прерывания ISA. А прерывания от PCI устройств стали выводить на линии IRQ 16-23. Теперь можно было не задумываться о конфликтах прерываний от ISA и PCI устройств. Также благодаря увеличенному количеству свободных линий прерываний возможно стало также увеличить количество линий PIRQx.  
  
  
  
Программирование I/O APIC и LAPIC осуществляется через MMIO. Регистры LAPIC расположены обычно по адресу 0xFEE00000, регистры I/O APIC по адресу 0xFEС00000. Хотя в принципе все эти адреса возможно переконфигурировать.  
  
Как и в случае с PIC первоначально отдельные микросхемы позже вошли в состав чипсета.  
  
В дальнейшем архитектура APIC получила модернизацию и новый вариант получил название xAPIC (x — extended). Сохранена обратная совместимость с предыдущим вариантом. Количество возможных CPU в системе увеличилось до 256.  
  
Следующий виток развития архитектуры получил название [x2APIC](https://www.intel.com/content/dam/doc/specification-update/64-architecture-x2apic-specification.pdf). Количество возможных CPU в системе увеличилось до 2^32. Контроллеры могут работать в режиме совместимости с xAPIC, а могут в новом режиме x2APIC, где программирование LAPIC осуществляется не через MMIO, а через MSR регистры (что гораздо быстрее). Cудя [по этой ссылке](https://blahg.josefsipek.net/?tag=xAPIC) для работы этого режима необходима поддержка IOMMU.  
  
Следует заметить, что в системе может быть несколько контроллеров I/O APIC. Например один на 24 прерывания в южном мосту, другой на 32 в северном. В контексте I/O APIC прерывания часто обозначаются GSI (Global System Interrupt). Так вот в такой системе будут GSI 0-55.  
  
Есть ли в CPU встроенный LAPIC и какой именно архитектуры можно понять по бит-флагам в CPUID.  
Чтобы система могла обнаружить LAPIC и I/O APIC, BIOS должен представить информацию о них системе либо через таблицу MPtable (старый метод), либо через таблицу ACPI (таблицу MADT в данном случае). Помимо общей информации, и в MPtable и в ACPI (на этот раз в таблице DSDT) должна содержаться информация о роутинге прерываний, то есть информация о том, какое устройство сидит на какой линии прерываний (аналог таблицы $PIR).  
  
О таблице MPTable можно почитать в официальной [спецификации](https://web.archive.org/web/20121002210153/http:/download.intel.com/design/archives/processors/pro/docs/24201606.pdf). Раньше спецификация была на сайте Intel, а сейчас её можно найти только в архиве. Спецификация ACPI сейчас расположена на сайте UEFI (текущая версия [6.2](https://www.uefi.org/sites/default/files/resources/ACPI_6_2.pdf)). Следует отметить, что с помощью ACPI можно указать роутинг прерываний и для систем без APIC (вместо использования таблицы $PIR).

MSI  
  
Предыдущий вариант с APIC хорош, но не лишён недостатков. Все эти линии прерываний от устройств усложняют схему, и увеличивают вероятности ошибок. На смену шины PCI пришёл PCI express, в котором линии прерываний решили просто-напросто убрать. Чтобы сохранить совместимость, сигналы о возникновении прерываний (INTx#) эмулируются отдельными видами сообщений. В этой схеме логическое сложение линий прерываний, которое раньше производилось физическим соединением проводов, легло на плечи PCI мостов. Однако поддержка legacy INTx прерываний — это лишь поддержка обратной совместимости с шиной PCI. На деле PCI express предложил новый метод доставки сообщений о прерываниях — MSI (Message Signaled Interrupts). В этом методе для сигнализации о прерывании устройство просто производит запись в MMIO область отведённую под LAPIC процессора.  
  
  
  
Если раньше на одно PCI устройство (то есть на все его функции) выделялось всего 4 прерывания, то сейчас сейчас стало возможным адресовать до 32 прерываний.  
  
В случае с MSI нет никакого sharing для линий, каждое прерывание соответствует своему устройству.  
  
Прерывания MSI решают также ещё одну проблему. Допустим устройство проводит memory-write транзакцию, и хочет сообщить о её завершении через прерывание. Но write транзакция может быть задержана на шине в процессе передачи (о чём устройство никак не знает), и сигнал о прерывании придёт до процессора раньше. Таким образом CPU будет читать ещё невалидные данные. В случае если используется MSI, информация об MSI передаётся также как и данные, и раньше прийти просто не сможет.  
  
Следует заметить, что прерывания MSI не могут работать без LAPIC, но использование MSI может заменить нам I/O APIC (упрощение дизайна).  
  
В последствии данный метод получил расширение MSI-X. Теперь каждое устройство может иметь до 2048 прерываний. И стало возможным указывать индивидуально каждому прерыванию на каком процессоре оно должно выполняться. Это может быть очень полезно для высоконагруженных устройств, например сетевых карт.  
  
Для поддержки MSI не требуется никаких дополнительных таблиц BIOS. Но устройство должно сообщить о поддержке MSI в одной из Capability в своём PCI Config, а драйвер устройства должен поддерживать работу с MSI.

Заключение

В данной статье мы рассмотрели эволюцию контроллеров прерываний, и получили общую теоретическую информацию о доставке прерываний от внешних устройств в x86 системе.  
  
В [следующей части](https://habr.com/post/431372/) мы посмотрим как на практике задействовать в Linux каждый из описанных контроллеров.

Ссылки:

* [Interrupt Controllers (Stuff in the Middle)](https://people.freebsd.org/~jhb/papers/bsdcan/2007/article/node4.html)
* [What do the different interrupts in PCIe do?](https://electronics.stackexchange.com/questions/76867/what-do-the-different-interrupts-in-pcie-do-i-referring-to-msi-msi-x-and-intx)
* [Reducing Interrupt Latency Through the Use of Message Signaled Interrupts](https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/msg-signaled-interrupts-paper.pdf)
* [Архитектура обработки прерываний в Intel совместимых системах](https://dev64.wordpress.com/2012/05/23/interrupt-subsystem-architecture/)

**Теги:**

* [apic](https://habr.com/ru/search/?q=%5Bapic%5D&target_type=posts)
* [pic](https://habr.com/ru/search/?q=%5Bpic%5D&target_type=posts)
* [msi](https://habr.com/ru/search/?q=%5Bmsi%5D&target_type=posts)
* [acpi](https://habr.com/ru/search/?q=%5Bacpi%5D&target_type=posts)
* [bios](https://habr.com/ru/search/?q=%5Bbios%5D&target_type=posts)

**Прерывания от внешних устройств в системе x86. Часть 2. Опции загрузки ядра Linux**

* [Настройка Linux](https://habr.com/ru/hub/linux/),
* [Системное программирование](https://habr.com/ru/hub/system_programming/)

В [предыдущей части](https://habr.com/post/430548/) мы рассмотрели эволюцию доставки прерываний от устройств в x86 системах (PIC → APIC → MSI), общую теорию и все необходимые термины.   
  
В этой практической части мы рассмотрим как откатиться к использованию устаревших методов доставки прерываний в Linux, а именно рассмотрим опции загрузки ядра:

* pci=nomsi
* noapic
* nolapic

Также мы посмотрим на порядок, в котором ОС смотрит таблицы роутинга прерываний (ACPI/MPtable/$PIR) и какое влияние на него окажет добавление опций загрузки:

* pci=noacpi
* acpi=noirq
* acpi=off

Возможно вы пробовали комбинации из всех этих опций, когда какое-либо устройство не работало из-за проблемы с прерываниями. Разберём, что именно они делают и как они меняют вывод /proc/interrupts.

Загрузка без дополнительных опций

Смотреть прерывания в данной статье мы будем на кастомной плате с Intel Haswell i7 с чипсетом lynxPoint-LP на которой запущен [coreboot](https://www.coreboot.org/).  
  
Информацию о прерываниях мы будем выводить через команду 

cat /proc/interrupts

Вывод при загрузке без дополнительных опций:

CPU0 CPU1 CPU2 CPU3

0: 15 0 0 0 IO-APIC-edge timer

1: 0 1 0 1 IO-APIC-edge i8042

8: 0 0 0 1 IO-APIC-edge rtc0

9: 0 0 0 0 IO-APIC-fasteoi acpi

12: 0 0 0 1 IO-APIC-edge

23: 16 247 7 10 IO-APIC-fasteoi ehci\_hcd:usb1

56: 0 0 0 0 PCI-MSI-edge aerdrv,PCIe PME

57: 0 0 0 0 PCI-MSI-edge aerdrv,PCIe PME

58: 0 0 0 0 PCI-MSI-edge aerdrv,PCIe PME

59: 0 0 0 0 PCI-MSI-edge aerdrv,PCIe PME

60: 0 0 0 0 PCI-MSI-edge aerdrv,PCIe PME

61: 0 0 0 0 PCI-MSI-edge aerdrv,PCIe PME

62: 3118 1984 972 3454 PCI-MSI-edge ahci

63: 1 0 0 0 PCI-MSI-edge eth59

64: 2095 57 4 832 PCI-MSI-edge eth59-rx-0

65: 6 18 1 1309 PCI-MSI-edge eth59-rx-1

66: 13 512 2 1 PCI-MSI-edge eth59-rx-2

67: 10 61 232 2 PCI-MSI-edge eth59-rx-3

68: 169 0 0 0 PCI-MSI-edge eth59-tx-0

69: 14 14 4 205 PCI-MSI-edge eth59-tx-1

70: 11 491 3 0 PCI-MSI-edge eth59-tx-2

71: 20 19 134 50 PCI-MSI-edge eth59-tx-3

72: 0 0 0 0 PCI-MSI-edge eth58

73: 2 1 0 152 PCI-MSI-edge eth58-rx-0

74: 3 150 2 0 PCI-MSI-edge eth58-rx-1

75: 2 34 117 2 PCI-MSI-edge eth58-rx-2

76: 153 0 2 0 PCI-MSI-edge eth58-rx-3

77: 4 0 2 149 PCI-MSI-edge eth58-tx-0

78: 4 149 2 0 PCI-MSI-edge eth58-tx-1

79: 4 0 117 34 PCI-MSI-edge eth58-tx-2

80: 153 0 2 0 PCI-MSI-edge eth58-tx-3

81: 66 106 2 101 PCI-MSI-edge snd\_hda\_intel

82: 928 5657 262 224 PCI-MSI-edge i915

83: 545 56 32 15 PCI-MSI-edge snd\_hda\_intel

NMI: 0 0 0 0 Non-maskable interrupts

LOC: 4193 3644 3326 3499 Local timer interrupts

SPU: 0 0 0 0 Spurious interrupts

PMI: 0 0 0 0 Performance monitoring interrupts

IWI: 290 233 590 111 IRQ work interrupts

RTR: 3 0 0 0 APIC ICR read retries

RES: 1339 2163 2404 1946 Rescheduling interrupts

CAL: 607 537 475 559 Function call interrupts

TLB: 163 202 164 251 TLB shootdowns

TRM: 48 48 48 48 Thermal event interrupts

THR: 0 0 0 0 Threshold APIC interrupts

MCE: 0 0 0 0 Machine check exceptions

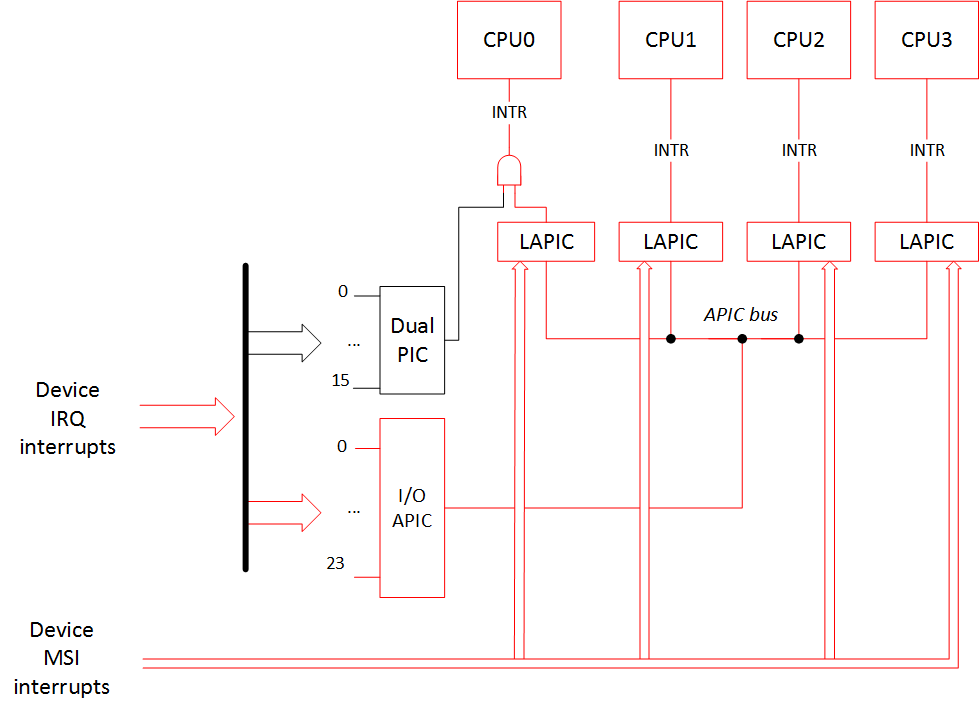
MCP: 3 3 3 3 Machine check polls

ERR: 0

MIS: 0

Файл /proc/interrupts предоставляет таблицу о количестве прерываний на каждом из процессоров в следующем виде:

* Первая колонка: номер прерывания
* Колонки CPUx: счётчики прерываний на каждом из процессоров
* Следующая колонка: вид прерывания:
  + IO-APIC-edge — прерывание по фронту на контроллер I/O APIC
  + IO-APIC-fasteoi — прерывание по уровню на контроллер I/O APIC
  + PCI-MSI-edge — MSI прерывание
  + XT-PIC-XT-PIC — прерывание на PIC контроллер (увидим позже)
* Последняя колонка: устройство, ассоциированное с данным прерыванием

Так вот, как и положено в современной системе, для устройств и драйверов, поддерживающих прерывания MSI/MSI-X, используются именно они. Остальные прерывания роутятся через I/O APIC.  
  
Упрощённо схему роутинга прерываний можно нарисовать так (красным помечены активные пути, чёрным неиспользуемые).  
  
  
  
Поддержка MSI/MSI-X устройством должна быть обозначена как соответствующая Capability в его конфигурационном пространстве PCI.  
  
В подтверждении приведём небольшой фрагмент вывода lspci для устройств, для которых обозначено, что они используют MSI/MSI-X. В нашем случае это SATA контроллер (прерывание ahci), 2 ethernet контроллера (прерывания eth58\* и eth59\*), графический контроллер (i915) и 2 контроллера HD Audio (snd\_hda\_intel).

lspci -v

00:02.0 VGA compatible controller: Intel Corporation Haswell-ULT Integrated Graphics Controller (rev 09) (prog-if 00 [VGA controller])

...

Capabilities: [90] MSI: Enable+ Count=1/1 Maskable- 64bit-

Capabilities: [d0] Power Management version 2

Capabilities: [a4] PCI Advanced Features

Kernel driver in use: i915

00:03.0 Audio device: Intel Corporation Haswell-ULT HD Audio Controller (rev 09

...

Capabilities: [60] MSI: Enable+ Count=1/1 Maskable- 64bit-

Capabilities: [70] Express Root Complex Integrated Endpoint, MSI 00

Kernel driver in use: snd\_hda\_intel

00:1b.0 Audio device: Intel Corporation 8 Series HD Audio Controller (rev 04)

...

Capabilities: [60] MSI: Enable+ Count=1/1 Maskable- 64bit+

Capabilities: [70] Express Root Complex Integrated Endpoint, MSI 00

Capabilities: [100] Virtual Channel

Kernel driver in use: snd\_hda\_intel

00:1f.2 SATA controller: Intel Corporation 8 Series SATA Controller 1 [AHCI mode] (rev 04) (prog-if 01 [AHCI 1.0])

...

Capabilities: [80] MSI: Enable+ Count=1/1 Maskable- 64bit-

Capabilities: [70] Power Management version 3

Capabilities: [a8] SATA HBA v1.0

Kernel driver in use: ahci

05:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)

...

Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+

Capabilities: [70] MSI-X: Enable+ Count=10 Masked-

Capabilities: [a0] Express Endpoint, MSI 00

Kernel driver in use: igb

05:00.1 Ethernet controller: Intel Corporation I350 Gigabit Network Connection (rev 01)

...

Capabilities: [50] MSI: Enable- Count=1/1 Maskable+ 64bit+

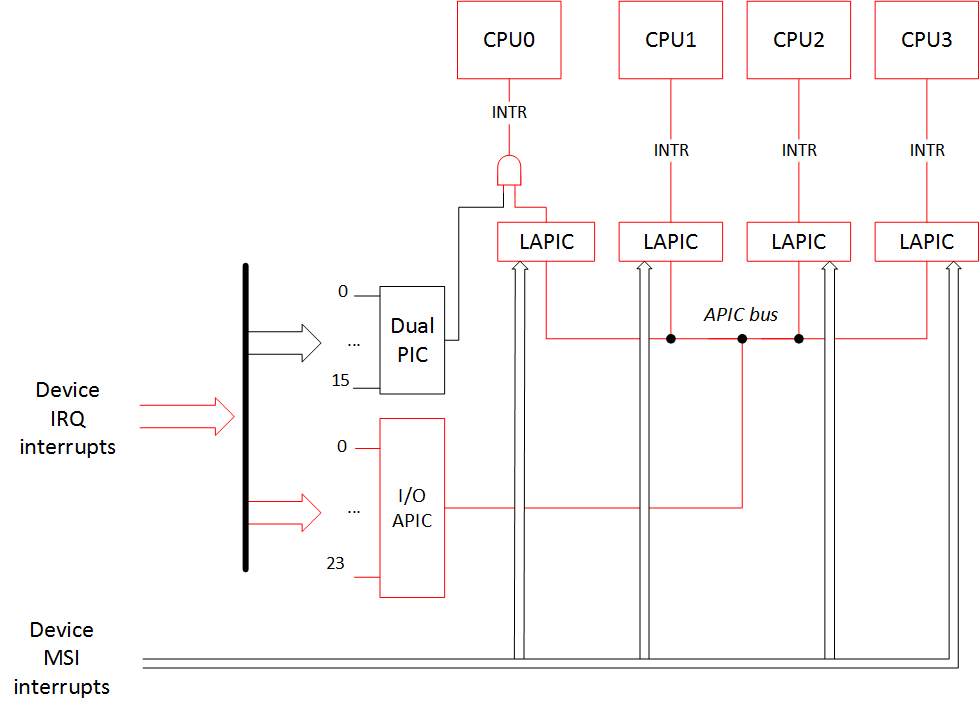
Capabilities: [70] MSI-X: Enable+ Count=10 Masked-

Capabilities: [a0] Express Endpoint, MSI 00

Kernel driver in use: igb

Как мы видим, у этих устройств присутствует строка либо «MSI: Enable+», либо «MSI-X: Enable+»  
  
Начнём деградировать систему. Для начала загрузимся с опцией pci=nomsi.

pci=nomsi

Благодаря этой опции MSI прерывания станут IO-APIC/XT-PIC в зависимости от используемого контроллера прерываний  
  
В данном случае у нас всё ещё приоритетный контроллер прерываний APIC, так что картина будет такая:  
  
  
  
Вывод /proc/interrupts:

CPU0 CPU1 CPU2 CPU3

0: 15 0 0 0 IO-APIC-edge timer

1: 0 1 0 1 IO-APIC-edge i8042

8: 0 0 1 0 IO-APIC-edge rtc0

9: 0 0 0 0 IO-APIC-fasteoi acpi

12: 0 0 0 1 IO-APIC-edge

16: 1314 5625 342 555 IO-APIC-fasteoi i915, snd\_hda\_intel, eth59

17: 5 0 1 34 IO-APIC-fasteoi eth58

21: 2882 2558 963 2088 IO-APIC-fasteoi ahci

22: 26 81 2 170 IO-APIC-fasteoi snd\_hda\_intel

23: 23 369 8 8 IO-APIC-fasteoi ehci\_hcd:usb1

NMI: 0 0 0 0 Non-maskable interrupts

LOC: 3011 3331 2435 2617 Local timer interrupts

SPU: 0 0 0 0 Spurious interrupts

PMI: 0 0 0 0 Performance monitoring interrupts

IWI: 197 228 544 85 IRQ work interrupts

RTR: 3 0 0 0 APIC ICR read retries

RES: 1708 2349 1821 1569 Rescheduling interrupts

CAL: 520 554 509 555 Function call interrupts

TLB: 187 181 205 179 TLB shootdowns

TRM: 102 102 102 102 Thermal event interrupts

THR: 0 0 0 0 Threshold APIC interrupts

MCE: 0 0 0 0 Machine check exceptions

MCP: 2 2 2 2 Machine check polls

ERR: 0

MIS: 0

Все прерывания MSI/MSI-X ожидаемо исчезли. Вместо них устройства теперь используют прерывания вида IO-APIC-fasteoi.  
  
Обратим внимание на то, что раньше до включения этой опции у eth58 и eth59 было по 9 прерываний! А сейчас только по одному. Ведь как мы помним, без MSI одной функции PCI доступно только одно прерывание!  
  
Немного информации из dmesg по инициализации ethernet контроллеров:  
  
— загрузка без опции pci=nomsi:

igb: Intel(R) Gigabit Ethernet Network Driver - version 5.0.5-k

igb: Copyright (c) 2007-2013 Intel Corporation.

acpi:acpi\_pci\_irq\_enable: igb 0000:05:00.0: PCI INT A -> GSI 16 (level, low) -> IRQ 16

igb 0000:05:00.0: irq 63 for MSI/MSI-X

igb 0000:05:00.0: irq 64 for MSI/MSI-X

igb 0000:05:00.0: irq 65 for MSI/MSI-X

igb 0000:05:00.0: irq 66 for MSI/MSI-X

igb 0000:05:00.0: irq 67 for MSI/MSI-X

igb 0000:05:00.0: irq 68 for MSI/MSI-X

igb 0000:05:00.0: irq 69 for MSI/MSI-X

igb 0000:05:00.0: irq 70 for MSI/MSI-X

igb 0000:05:00.0: irq 71 for MSI/MSI-X

igb 0000:05:00.0: irq 63 for MSI/MSI-X

igb 0000:05:00.0: irq 64 for MSI/MSI-X

igb 0000:05:00.0: irq 65 for MSI/MSI-X

igb 0000:05:00.0: irq 66 for MSI/MSI-X

igb 0000:05:00.0: irq 67 for MSI/MSI-X

igb 0000:05:00.0: irq 68 for MSI/MSI-X

igb 0000:05:00.0: irq 69 for MSI/MSI-X

igb 0000:05:00.0: irq 70 for MSI/MSI-X

igb 0000:05:00.0: irq 71 for MSI/MSI-X

igb 0000:05:00.0: added PHC on eth0

igb 0000:05:00.0: Intel(R) Gigabit Ethernet Network Connection

igb 0000:05:00.0: eth0: (PCIe:5.0Gb/s:Width x1) 00:15:d5:03:00:2a

igb 0000:05:00.0: eth0: PBA No: 106300-000

igb 0000:05:00.0: Using MSI-X interrupts. 4 rx queue(s), 4 tx queue(s)

acpi:acpi\_pci\_irq\_enable: igb 0000:05:00.1: PCI INT B -> GSI 17 (level, low) -> IRQ 17

igb 0000:05:00.1: irq 72 for MSI/MSI-X

igb 0000:05:00.1: irq 73 for MSI/MSI-X

igb 0000:05:00.1: irq 74 for MSI/MSI-X

igb 0000:05:00.1: irq 75 for MSI/MSI-X

igb 0000:05:00.1: irq 76 for MSI/MSI-X

igb 0000:05:00.1: irq 77 for MSI/MSI-X

igb 0000:05:00.1: irq 78 for MSI/MSI-X

igb 0000:05:00.1: irq 79 for MSI/MSI-X

igb 0000:05:00.1: irq 80 for MSI/MSI-X

igb 0000:05:00.1: irq 72 for MSI/MSI-X

igb 0000:05:00.1: irq 73 for MSI/MSI-X

igb 0000:05:00.1: irq 74 for MSI/MSI-X

igb 0000:05:00.1: irq 75 for MSI/MSI-X

igb 0000:05:00.1: irq 76 for MSI/MSI-X

igb 0000:05:00.1: irq 77 for MSI/MSI-X

igb 0000:05:00.1: irq 78 for MSI/MSI-X

igb 0000:05:00.1: irq 79 for MSI/MSI-X

igb 0000:05:00.1: irq 80 for MSI/MSI-X

igb 0000:05:00.1: added PHC on eth1

igb 0000:05:00.1: Intel(R) Gigabit Ethernet Network Connection

igb 0000:05:00.1: eth1: (PCIe:5.0Gb/s:Width x1) 00:15:d5:03:00:2b

igb 0000:05:00.1: eth1: PBA No: 106300-000

igb 0000:05:00.1: Using MSI-X interrupts. 4 rx queue(s), 4 tx queue(s)

— загрузка с опцией pci=nomsi

igb: Intel(R) Gigabit Ethernet Network Driver - version 5.0.5-k

igb: Copyright (c) 2007-2013 Intel Corporation.

acpi:acpi\_pci\_irq\_enable: igb 0000:05:00.0: PCI INT A -> GSI 16 (level, low) -> IRQ 16

igb 0000:05:00.0: added PHC on eth0

igb 0000:05:00.0: Intel(R) Gigabit Ethernet Network Connection

igb 0000:05:00.0: eth0: (PCIe:5.0Gb/s:Width x1) 00:15:d5:03:00:2a

igb 0000:05:00.0: eth0: PBA No: 106300-000

igb 0000:05:00.0: Using legacy interrupts. 1 rx queue(s), 1 tx queue(s)

acpi:acpi\_pci\_irq\_enable: igb 0000:05:00.1: PCI INT B -> GSI 17 (level, low) -> IRQ 17

igb 0000:05:00.1: added PHC on eth1

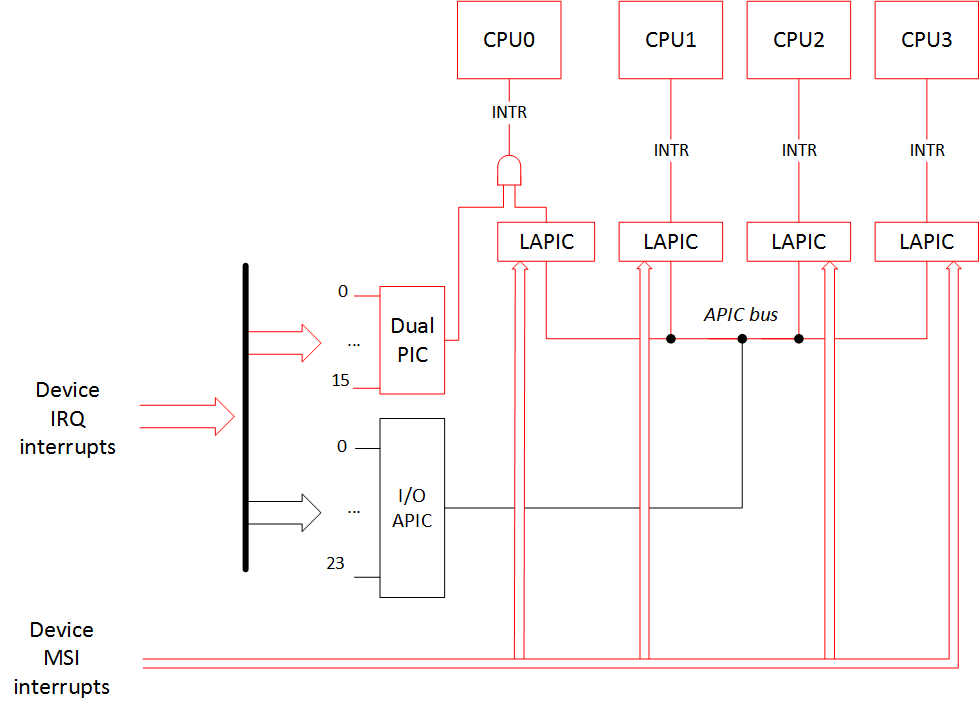
igb 0000:05:00.1: Intel(R) Gigabit Ethernet Network Connection

igb 0000:05:00.1: eth1: (PCIe:5.0Gb/s:Width x1) 00:15:d5:03:00:2b

igb 0000:05:00.1: eth1: PBA No: 106300-000

igb 0000:05:00.1: Using legacy interrupts. 1 rx queue(s), 1 tx queue(s)

Из-за уменьшения количества прерываний на устройство, включение данной опции может приводить к существенному ограничению производительности работы драйвера (это без учёта того, что согласно исследованию Intel [Reducing Interrupt Latency Through the Use of Message Signaled Interrupts](https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/msg-signaled-interrupts-paper.pdf)прерывания через MSI в 3 раза быстрее чем через IO-APIC и в 5 раз быстрее чем через PIC).

noapic  
Данная опция отключает I/O APIC. MSI прерывания всё ещё могут идти на все CPU, но прерывания от устройств смогут идти только на CPU0, так как PIC связан только с CPU0. Но LAPIC работает и другие CPU могут работать и обрабатывать прерывания.  
  


CPU0 CPU1 CPU2 CPU3

0: 5 0 0 0 XT-PIC-XT-PIC timer

1: 2 0 0 0 XT-PIC-XT-PIC i8042

2: 0 0 0 0 XT-PIC-XT-PIC cascade

8: 1 0 0 0 XT-PIC-XT-PIC rtc0

9: 0 0 0 0 XT-PIC-XT-PIC acpi

12: 172 0 0 0 XT-PIC-XT-PIC ehci\_hcd:usb1

56: 0 0 0 0 PCI-MSI-edge aerdrv, PCIe PME

57: 0 0 0 0 PCI-MSI-edge aerdrv, PCIe PME

58: 0 0 0 0 PCI-MSI-edge aerdrv, PCIe PME

59: 0 0 0 0 PCI-MSI-edge aerdrv, PCIe PME

60: 0 0 0 0 PCI-MSI-edge aerdrv, PCIe PME

61: 0 0 0 0 PCI-MSI-edge aerdrv, PCIe PME

62: 2833 2989 1021 811 PCI-MSI-edge ahci

63: 0 1 0 0 PCI-MSI-edge eth59

64: 301 52 9 3 PCI-MSI-edge eth59-rx-0

65: 12 24 3 178 PCI-MSI-edge eth59-rx-1

66: 14 85 6 2 PCI-MSI-edge eth59-rx-2

67: 17 24 307 1 PCI-MSI-edge eth59-rx-3

68: 70 18 8 10 PCI-MSI-edge eth59-tx-0

69: 7 0 0 23 PCI-MSI-edge eth59-tx-1

70: 15 227 2 2 PCI-MSI-edge eth59-tx-2

71: 18 6 27 2 PCI-MSI-edge eth59-tx-3

72: 0 0 0 0 PCI-MSI-edge eth58

73: 1 0 0 27 PCI-MSI-edge eth58-rx-0

74: 1 22 0 5 PCI-MSI-edge eth58-rx-1

75: 1 0 22 5 PCI-MSI-edge eth58-rx-2

76: 23 0 0 5 PCI-MSI-edge eth58-rx-3

77: 1 0 0 27 PCI-MSI-edge eth58-tx-0

78: 1 22 0 5 PCI-MSI-edge eth58-tx-1

79: 1 0 22 5 PCI-MSI-edge eth58-tx-2

80: 23 0 0 5 PCI-MSI-edge eth58-tx-3

81: 187 17 70 7 PCI-MSI-edge snd\_hda\_intel

82: 698 1647 247 129 PCI-MSI-edge i915

83: 438 135 16 59 PCI-MSI-edge snd\_hda\_intel

NMI: 0 0 0 0 Non-maskable interrupts

LOC: 1975 2499 2245 1474 Local timer interrupts

SPU: 0 0 0 0 Spurious interrupts

PMI: 0 0 0 0 Performance monitoring interrupts

IWI: 132 67 429 91 IRQ work interrupts

RTR: 3 0 0 0 APIC ICR read retries

RES: 1697 2178 1903 1541 Rescheduling interrupts

CAL: 561 496 534 567 Function call interrupts

TLB: 229 254 170 137 TLB shootdowns

TRM: 78 78 78 78 Thermal event interrupts

THR: 0 0 0 0 Threshold APIC interrupts

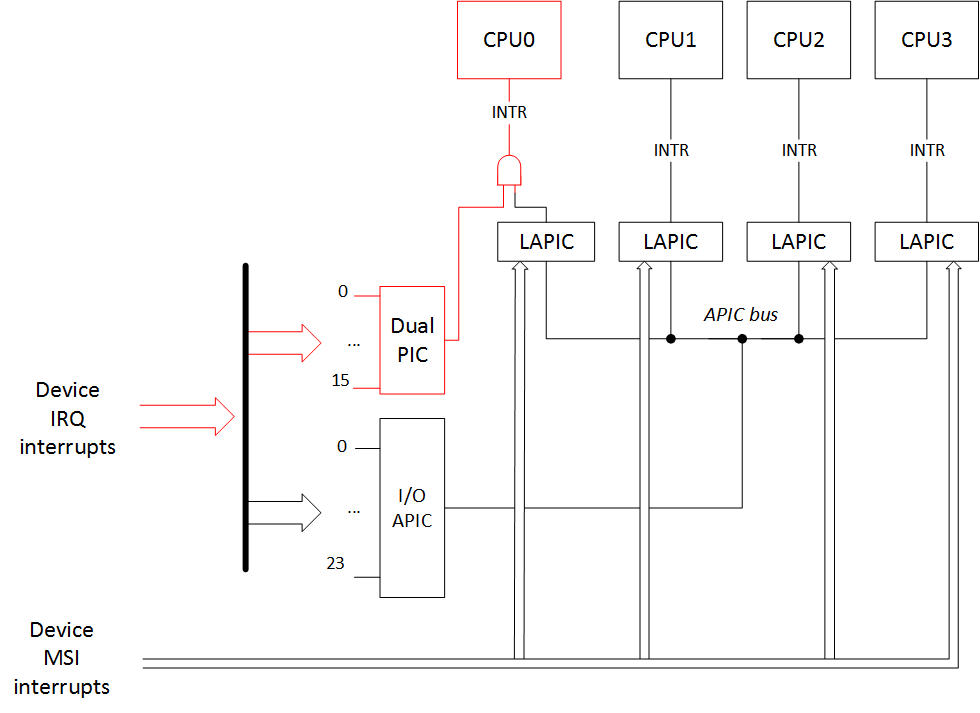
MCE: 0 0 0 0 Machine check exceptions

MCP: 2 2 2 2 Machine check polls

ERR: 0

MIS: 0

Как видим, все прерывания IO-APIC-\* превратились в XT-PIC-XT-PIC, причём эти прерывания роутятся только на CPU0. Прерывания MSI остались без изменений и идут на все CPU0-3.

nolapic  
Отключает LAPIC. MSI прерывания не могут работать без LAPIC, I/O APIC не может работать без LAPIC. Поэтому все прерывания от устройств будут идти на PIC, а он работает только с CPU0. И без LAPIC остальные CPU даже работать в системе не будут.  
  
  
  
Вывод /proc/interrupts:

CPU0

0: 6416 XT-PIC-XT-PIC timer

1: 2 XT-PIC-XT-PIC i8042

2: 0 XT-PIC-XT-PIC cascade

3: 5067 XT-PIC-XT-PIC aerdrv, aerdrv, PCIe PME, PCIe PME, i915, snd\_hda\_intel, eth59

4: 32 XT-PIC-XT-PIC aerdrv, aerdrv, PCIe PME, PCIe PME, eth58

5: 0 XT-PIC-XT-PIC aerdrv, PCIe PME

6: 0 XT-PIC-XT-PIC aerdrv, PCIe PME

8: 1 XT-PIC-XT-PIC rtc0

9: 0 XT-PIC-XT-PIC acpi

11: 274 XT-PIC-XT-PIC snd\_hda\_intel

12: 202 XT-PIC-XT-PIC ehci\_hcd:usb1

15: 7903 XT-PIC-XT-PIC ahci

NMI: 0 Non-maskable interrupts

LOC: 0 Local timer interrupts

SPU: 0 Spurious interrupts

PMI: 0 Performance monitoring interrupts

IWI: 0 IRQ work interrupts

RTR: 0 APIC ICR read retries

RES: 0 Rescheduling interrupts

CAL: 0 Function call interrupts

TLB: 0 TLB shootdowns

TRM: 0 Thermal event interrupts

THR: 0 Threshold APIC interrupts

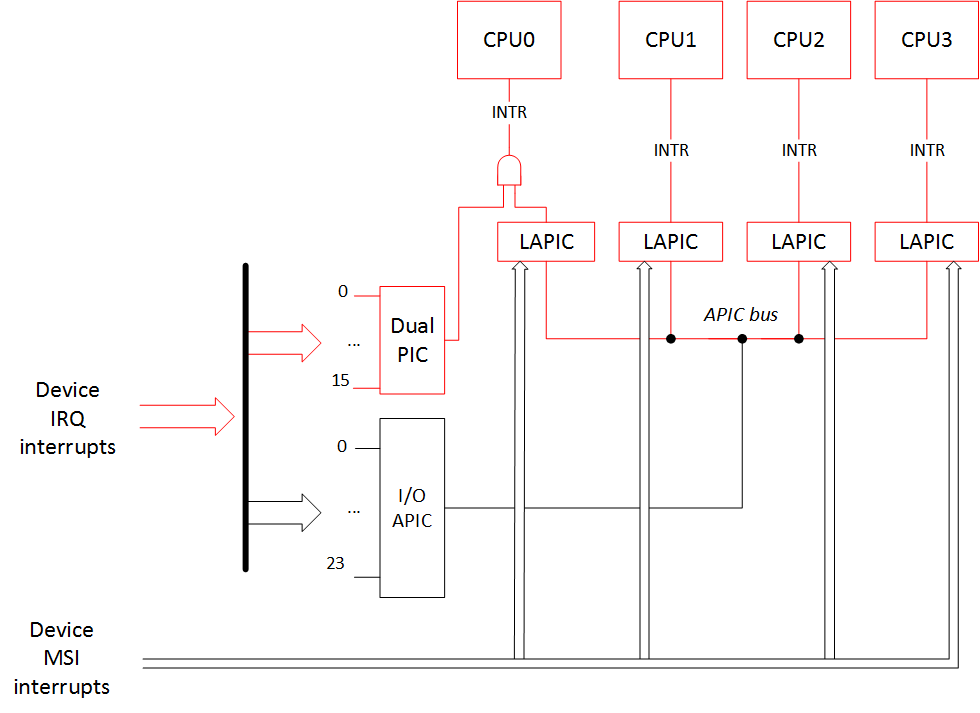
MCE: 0 Machine check exceptions

MCP: 1 Machine check polls

ERR: 0

MIS: 0

Комбинации:

На самом деле всего одна для нового варианта: «noapic pci=nomsi». Все прерывания от устройств смогут идти только на CPU0 через PIC. Но LAPIC работает и другие CPU могут работать и обрабатывать прерывания.  
Одна, потому что с «nolapic» можно ничего не комбинировать, т.к. эта опция и так сделает недоступным I/O APIC и MSI. Так что если вы когда-то прописывали опции загрузки «noapic nolapic» (или самый распространённый вариант «acpi=off noapic nolapic»), то судя по всему вы набирали лишние буквы.   
Итак, что будет от опций «noapic pci=nomsi»:   
  
  
  
Вывод /proc/interrupts:

CPU0 CPU1 CPU2 CPU3

0: 5 0 0 0 XT-PIC-XT-PIC timer

1: 2 0 0 0 XT-PIC-XT-PIC i8042

2: 0 0 0 0 XT-PIC-XT-PIC cascade

3: 5072 0 0 0 XT-PIC-XT-PIC i915, snd\_hda\_intel, eth59

4: 32 0 0 0 XT-PIC-XT-PIC eth58

8: 1 0 0 0 XT-PIC-XT-PIC rtc0

9: 0 0 0 0 XT-PIC-XT-PIC acpi

11: 281 0 0 0 XT-PIC-XT-PIC snd\_hda\_intel

12: 200 0 0 0 XT-PIC-XT-PIC ehci\_hcd:usb1

15: 7930 0 0 0 XT-PIC-XT-PIC ahci

NMI: 0 0 0 0 Non-maskable interrupts

LOC: 2595 2387 2129 1697 Local timer interrupts

SPU: 0 0 0 0 Spurious interrupts

PMI: 0 0 0 0 Performance monitoring interrupts

IWI: 159 90 482 135 IRQ work interrupts

RTR: 3 0 0 0 APIC ICR read retries

RES: 1568 1666 1810 1833 Rescheduling interrupts

CAL: 431 556 549 558 Function call interrupts

TLB: 124 184 156 274 TLB shootdowns

TRM: 116 116 116 116 Thermal event interrupts

THR: 0 0 0 0 Threshold APIC interrupts

MCE: 0 0 0 0 Machine check exceptions

MCP: 2 2 2 2 Machine check polls

ERR: 0

MIS: 0

Таблицы роутинга прерываний и опции «acpi=noirq», «pci=noacpi», «acpi=off»

Как операционная система получает информацию о роутинге прерываний от устройств? BIOS подготавливает информацию для ОС в виде:

* ACPI таблиц (методы \_PIC/\_PRT)
* \_MP\_ таблицы (MPtable)
* $PIR таблицы
* Регистров 0x3C/0x3D конфигурационного пространства PCI устройств  
  Следует отметить, что для обозначения прерываний MSI BIOSу не надо ничего дополнительно делать, вся вышеупомянутая информация нужна только для линий APIC/PIC прерываний.  
    
  Таблицы в списке выше обозначены в порядке приоритета. Рассмотрим это подробней.  
    
  Допустим BIOS предоставил все эти данные и мы грузимся без каких-либо дополнительных опций:
* ОС находит таблицы ACPI
* ОС выполняет метод ACPI "\_PIC", передаёт ему аргумент, что нужно грузиться в режиме APIC. Тут код метода обычно сохраняет выбранный режим в переменной (допустим PICM=1)
* Для получения данных о прерываниях ОС вызывает метод ACPI "\_PRT". Он внутри себя проверяет переменную PICM и возвращает роутинг для APIC случая

В случае если мы грузимся с опцией **noapic**:

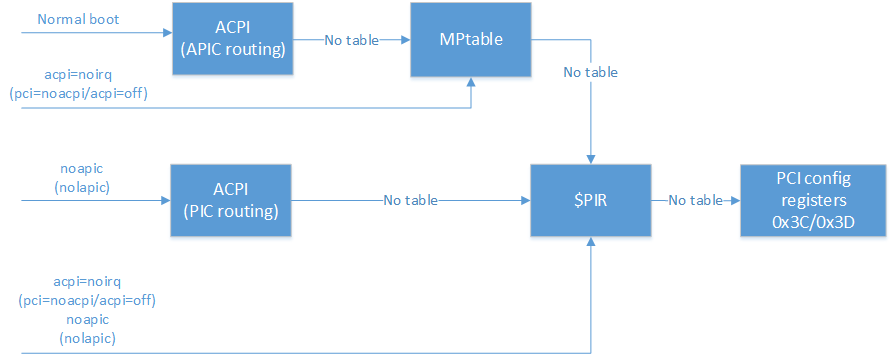
* ОС находит таблицы ACPI
* ОС выполняет метод ACPI "\_PIC", передаёт ему аргумент, что нужно грузиться в режиме PIC. Тут код метода обычно сохраняет выбранный режим в переменной (допустим PICM=0)
* Для получения данных о прерываниях ОС вызывает метод ACPI "\_PRT". Он внутри себя проверяет переменную PICM и возвращает роутинг для PIC случая

Если таблица ACPI отсутствует или функционал роутинга прерываний через ACPI отключен с помощью опций **acpi=noirq** или **pci=noacpi** (или ACPI полностью выключен с помощью **acpi=off**), то ОС смотрит для роутинга прерываний таблицу MPtable (\_MP\_):

* ОС не находит/не смотрит таблицы ACPI
* ОС находит таблицу MPtable (\_MP\_)

Если таблица ACPI отсутствует или функционал роутинга прерываний через ACPI отключен с помощью опций **acpi=noirq** или **pci=noacpi** (или ACPI полностью выключен с помощью **acpi=off**) и если таблица MPtable (\_MP\_) отсутствует (или передана опция загрузки **noapic** или **nolapic**):

* ОС не находит/не смотрит таблицу ACPI
* ОС не находит/не смотрит таблицу MPtable (\_MP\_)
* ОС находит таблицу $PIR

Если и таблицы $PIR нет, или она не полна, то операционная система для угадывания прерываний будет смотреть значения регистров 0x3C/0x3D конфигурационного пространства PCI устройств.  
  
Суммируем всё вышеизложенное следующей картинкой:  
  
  
  
Следует помнить, что не каждый BIOS предоставляет все 3 таблицы (ACPI/MPtable/$PIR), так что если вы передали опцию загрузчику отказаться от использования ACPI или ACPI и MPtable для роутинга прерываний, далеко не факт, что ваша система загрузится.  
  
**Замечание 1**: в случае если мы попытаемся загрузиться в режиме APIC с опцией acpi=noirq и без наличия MPtable, то картина прерываний будет как и в случае обычной загрузки с единственной опцией noapic. Операционная система сама перейдёт в режим PIC прерываний.  
В случае если мы попытаемся загрузиться вообще без таблиц ACPI (acpi=off) и не предоставив MPtable, то картина будет такая:

CPU0

0: 6 XT-PIC-XT-PIC timer

1: 2 XT-PIC-XT-PIC i8042

2: 0 XT-PIC-XT-PIC cascade

8: 0 XT-PIC-XT-PIC rtc0

12: 373 XT-PIC-XT-PIC ehci\_hcd:usb1

16: 0 PCI-MSI-edge PCIe PME

17: 0 PCI-MSI-edge PCIe PME

18: 0 PCI-MSI-edge PCIe PME

19: 0 PCI-MSI-edge PCIe PME

20: 0 PCI-MSI-edge PCIe PME

21: 0 PCI-MSI-edge PCIe PME

22: 8728 PCI-MSI-edge ahci

23: 1 PCI-MSI-edge eth59

24: 1301 PCI-MSI-edge eth59-rx-0

25: 113 PCI-MSI-edge eth59-tx-0

26: 0 PCI-MSI-edge eth58

27: 45 PCI-MSI-edge eth58-rx-0

28: 45 PCI-MSI-edge eth58-tx-0

29: 1280 PCI-MSI-edge snd\_hda\_intel

NMI: 2 Non-maskable interrupts

LOC: 24076 Local timer interrupts

SPU: 0 Spurious interrupts

PMI: 2 Performance monitoring interrupts

IWI: 2856 IRQ work interrupts

RTR: 0 APIC ICR read retries

RES: 0 Rescheduling interrupts

CAL: 0 Function call interrupts

TLB: 0 TLB shootdowns

TRM: 34 Thermal event interrupts

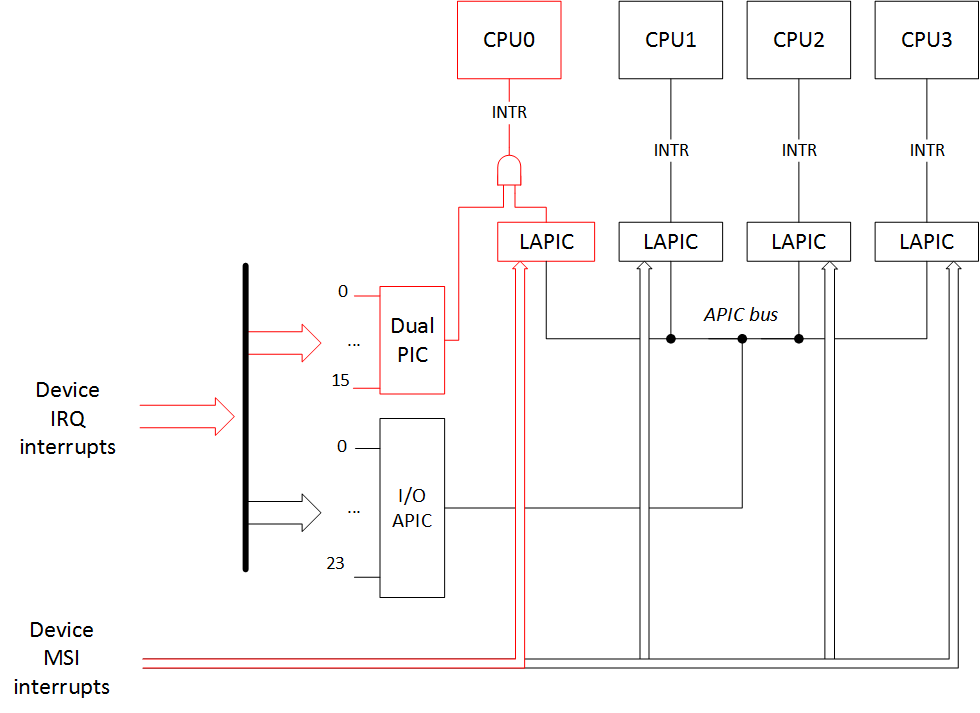
THR: 0 Threshold APIC interrupts

MCE: 0 Machine check exceptions

MCP: 2 Machine check polls

ERR: 0

MIS: 0

Это проиcходит из-за того, что без ACPI таблицы MADT ([Multiple APIC Description Table](https://wiki.osdev.org/MADT)) и необходимой информации из MPtable, операционная система не знает APIC идентификаторы (APIC ID) для других процессоров и не может с ними работать, но LAPIC основного процессора работает, так как мы это не запрещали, и MSI прерывания могут на него приходить. То есть будет так:  
  
  
  
**Замечание 2**: в целом роутинг прерываний при использовании ACPI в случае APIC совпадает с роутингом прерываний через MPtable. А роутинг прерываний через ACPI в случае использования PIC совпадает с роутингом прерываний через $PIR. Так что и выводы /proc/interrupts отличаться не должны. Однако в процессе исследований заметил одну странность. При роутинге через MPtable в выводе почему-то присутствует каскадное прерывание «XT-PIC-XT-PIC cascade».

CPU0 CPU1 CPU2 CPU3

0: 15 0 0 0 IO-APIC-edge timer

1: 2 0 0 0 IO-APIC-edge i8042

2: 0 0 0 0 XT-PIC-XT-PIC cascade

8: 0 1 0 0 IO-APIC-edge rtc0

9: 0 0 0 0 IO-APIC-edge acpi

...

Немного странно, что так происходит, но в [документации ядра](https://elixir.bootlin.com/linux/latest/source/Documentation/x86/i386/IO-APIC.txt) вроде говорится, что это нормально.

Заключение:

В заключении ещё раз обозначим разобранные опции.  
Опции выбора контроллера прерываний:

* **pci=nomsi** — MSI прерывания станут IO-APIC/XT-PIC в зависимости от используемого контроллера прерываний
* **noapic** — Отключает I/O APIC. MSI прерывания всё ещё могут идти на все CPU, остальные прерывания от устройств смогут идти только на PIC, а он работает только с CPU0. Но LAPIC работает и другие CPU могут работать и обрабатывать прерывания
* **noapic pci=nomsi** — Все прерывания от устройств могут идти только на PIC, а он работает только с CPU0. Но LAPIC работает и другие CPU могут работать и обрабатывать прерывания
* **nolapic** — Отключает LAPIC. MSI прерывания не могут работать без LAPIC, I/O APIC не может работать без LAPIC. Все прерывания от устройств будут идти на PIC, а он работает только с CPU0. И без LAPIC остальные CPU не будут работать.

Опции выбора приоритетной таблицы роутинга прерываний:

* **без опций** — роутинг через APIC с помощью таблиц ACPI
* **noapic** — роутинг через PIC с помощью таблиц ACPI
* **acpi=noirq** (**pci=noacpi**/**acpi=off**) — роутинг через APIC с помощью таблицы MPtable
* **acpi=noirq** (**pci=noacpi**/**acpi=off**) **noapic** (**nolapic**) — роутинг через PIC с помощью таблицы $PIR  
  В следующей части посмотрим как coreboot настраивает чипсет для роутинга прерываний.

**Теги:**

* [linux](https://habr.com/ru/search/?q=%5Blinux%5D&target_type=posts)
* [acpi](https://habr.com/ru/search/?q=%5Bacpi%5D&target_type=posts)
* [apic](https://habr.com/ru/search/?q=%5Bapic%5D&target_type=posts)
* [pic](https://habr.com/ru/search/?q=%5Bpic%5D&target_type=posts)
* [msi](https://habr.com/ru/search/?q=%5Bmsi%5D&target_type=posts)
* [bios](https://habr.com/ru/search/?q=%5Bbios%5D&target_type=posts)
* [irq](https://habr.com/ru/search/?q=%5Birq%5D&target_type=posts)
* [interrupts](https://habr.com/ru/search/?q=%5Binterrupts%5D&target_type=posts)