

ハイパーバイザの作り方～ちゃんと理解する仮想化技術～ 付属資料最近の PC アーキテクチャにおける割り込みルーティングの仕組み

はじめに

Linux における `/proc/irq/<IRQ>/smp_affinity` はハードウェアにどのような設定を行うことにより実現されているのか、或いは最近の PC アーキテクチャにおける割り込みの仕組みはどうなっているのか、という辺りが知りたかったので調べてみた。

結構こんがらがっているので、予想外に時間を食ってしまった...まだ調べ尽くせていないが、一旦現時点での理解を書いておこうと思う。

前提条件

1. 2 つ以上の CPU コアを持つ、Core2 世代或いは Core i シリーズ世代の Intel CPU / チップセット
2. 割り込みを行う主体は PCIe デバイスである（主に NIC を想定しているが、これに限定されない）
3. Legacy な 8259 割り込みコントローラを使うことは考慮しない
4. x86_64 向け Linux カーネル（解析に使っているバージョンは 3.2.0+）が動作している
5. 仮想化は使用しない

PCIe に於ける割り込みの種類

レガシー割り込み (INTx)

PCI 規格に最初から用意されていた割り込み方法で、大半の PCI デバイスはこの割り込みを用いている。PCI バスのメインラインとは別に用意された割り込み用の物理的なピンを用いて割り込みを通知する。

PCIe には割り込み用ピンは用意されておらず、帯域内メッセージを用いるレガシー割り込みエミュレーションによってソフト的な互換性を維持しているものの、基本的には MSI / MSI-X 割り込みへ移行することが推奨されているものと思われる。

MSI 割り込み

PCI 2.3 から追加された割り込みモードでピンを使用しない帯域内メッセージで割り込みを行う。デバイスあたり最大 32 個の MSI メッセージをサポートしている。

MSI-X 割り込み

PCI 3.0 ではオプションとされ PCIe 1.0 から必須とされた割り込みモードで、MSI 割り込みの拡張版。デバイスあたり最大 2048 個のメッセージをサポートしている。

割り込みルーティング

レガシー割り込み

デバイスからピン経由で割り込みを通知 IOAPIC で Redirection Table Entry を参照、通知先 LAPIC を決定 CPU 内の LAPIC へ割り込みを通知

MSI 割り込み

デバイスは PCI Configuration Space の Capability Structure 内の MSI フィールドを参照、MSI Address レジスタと MSI Data レジスタの値から通知先 LAPIC と LAPIC 上のベクタ番号を決定 CPU 内の LAPIC へ割り込みを通知

MSI-X 割り込み

レジスタの構成が異なる（ベクタ毎に Address と Data が用意されている）が基本的な仕組みは MSI 割り込みと同様

Capability Structure

（“BIOS が PCI Express を初期化する手順が見えてきた:なひたふ JTAG 日記”）を見るとイメージが分かると思うが、Configuration Space から Linked List 状に複数の capability が繋がる構造になっていて、CAPID が 0xd0 なのが MSI のフィールドで、ここには MSICTL, MSIAR, MSIDR の 3 つのレジスタがある。

MSI Control Register(MSICTL)

どの CPU に割り込むかを考える上では重要ではないので省略

MSI Address Register(MSIAR)

- 31:20 = 0xfee
- 19:12 = Destination ID
- 11:4 = IA32 では未使用
- 3 = Address Redirection Hint(RH)
 - 0: Directed
 - 1: Redirectable
- 2 = Address Destination Mode(DM)
 - 0: Physical Mode
 - 1: Logical Mode
- 1:0 = 予約

Destination Mode が Logical かつ Redirection Hint が Redirectable な場合は Destination ID でビットが立っている CPU の中で Task Priority Register(TPR) が最も低い CPU の LAPIC へ割り込みが送られる。それ以外の RH, DM の組み合わせでは Destination ID で指定されているビットの中で特定の CPU の LAPIC へ割り込みが送られる。

Physical Mode で Destination ID が 0xff の場合はブロードキャスト割り込みを行う。

MSI Data Register(MSIDR)

- 31:16 = 0x0000
- 15 = Trigger mode
 - 0: Edge
 - 1: Level
- 14 = Delivery status
 - 0: Deassert
 - 1: Assert
- 13:12 = 0x00
- 11:8 = Delivery mode
 - 0000: Fixed
 - 0001: Lowest priority
 - 0010: SMI/PMI/MCA
 - 0011: Reserved
 - 0100: NMI
 - 0101: INIT
 - 0110: Reserved
 - 0111: ExtINT

- 1000-1111: Reserved
- 7:0 = Interrupt Vector

Delivery mode が Fixed の場合は Destination に指定された全ての CPU へ割り込みを行う。Lowest Priority の場合は Task Priority Register の値が最も低い CPU へ割り込みを行う。Interrupt Vector に割り込み先 LAPIC の Vector 番号を指定。

Linux カーネルで実際にレジスタの値を設定している所をしてみる

msi_compose_msg^{*1}でレジスタに書き込みたい値を用意しているので、これを見してみる。msg->address_lo が MSIAR レジスタで、apic->irq_dest_mode が 0 なら physical mode、1 なら logical mode を設定、apic->irq_delivery_mode が dest_LowestPrio なら Redirectable(MSI_ADDR_REDIRECTION_LOWPRI) を、そうでなければ Directed(MSI_ADDR_REDIRECTION_CPU) を設定、変数 dest を Destination ID として設定している。

msg->data が MSIDR レジスタで、apic->irq_delivery_mode が dest_LowestPrio なら Lowest priority を、そうでなければ Fixed を設定、cfg->vector の値を Interrupt Vector として設定している。

apic->irq_dest_mode と apic->irq_delivery_mode の値は IO APIC のドライバ毎に違うのだが、x86_64 の標準ドライバの apic_flat_64.c^{*2}では irq_dest_mode は 1, irq_delivery_mode は dest_LowestPrio に設定されている。

これらの値は割り込み初期化時に設定され、/proc/irq/<IRQ>/smp_affinity の書き換え時にも維持される。smp_affinity の書き換え時には、Destination ID と Interrupt Vector だけが変更される^{*3}。

全ての環境で Logical mode かつ Lowest priority が使えるとは限らないので、場合によっては Physical Mode で初期化されていて smp_affinity の値を 0xff にしても CPU0 にしか割り込まないという挙動を行う事も有り得る。実際、論理 CPU が 12 個ある Core i7 上で Linux 3.2.0+ を走らせている環境では Extended Physical Mode で初期化されていて、割り込み分散が行われていなかった。

/proc/irq/<IRQ>/smp_affinity の書き換えで PCI コンフィグレーション空間はどのように書き換わるか

例えば Thinkpad x200 にはこんなデバイスがあります。(dmesg から抜粋)

```
e1000e: Intel(R) PRO/1000 Network Driver - 1.5.1-k
e1000e: Copyright(c) 1999 - 2011 Intel Corporation.
e1000e 0000:00:19.0: PCI INT A -> GSI 20 (level, low) -> IRQ 20
e1000e 0000:00:19.0: setting latency timer to 64
```

*1 http://lxr.linux.no/linux+v3.2/arch/x86/kernel/apic/io_apic.c#L3167

*2 http://lxr.linux.no/linux+v3.2/arch/x86/kernel/apic/apic_flat_64.c#L180

*3 http://lxr.linux.no/linux+v3.2/arch/x86/kernel/apic/io_apic.c#L3201

```
e1000e 0000:00:19.0: irq 44 for MSI/MSI-X
e1000e 0000:00:19.0: eth0: (PCI Express:2.5GT/s:Width x1) 00:1f:16:2a:a4:59
e1000e 0000:00:19.0: eth0: Intel(R) PRO/1000 Network Connection
e1000e 0000:00:19.0: eth0: MAC: 7, PHY: 8, PBA No: 1008FF-0FF
udev[16200]: renamed network interface eth0 to eth4
```

IRQ44 の MSI 割り込みを一つ持つ e1000e で、PCI のアドレスは 00:19.0 です。

```
# cat /proc/irq/44/smp_affinity
3
```

CPU は cpu0 と cpu1 なので、全ての CPU のビットを立ててるから 3。

```
# grep eth4 /proc/interrupts
44:      50037      49330  PCI-MSI-edge      eth4
```

設定通り、両側の CPU に割り込んでます。この時、MSI Address Register と MSI Data Register にはどのような値が設定されているか確認してみます。

```
# lspci -vvvv -s 00:19.0
00:19.0 Ethernet controller: Intel Corporation 82567LM Gigabit Network Connection (rev 03)
Subsystem: Lenovo Device 20ee
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR+ FastB2B- Dis
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- I
Latency: 0
Interrupt: pin A routed to IRQ 44
Region 0: Memory at f2600000 (32-bit, non-prefetchable) [size=128K]
Region 1: Memory at f2625000 (32-bit, non-prefetchable) [size=4K]
Region 2: I/O ports at 1840 [size=32]
Capabilities: [c8] Power Management version 2
Flags: PMEClk- DSI+ D1- D2- AuxCurrent=0mA PME(D0+,D1-,D2-,D3hot+,D3cold+)
Status: D0 NoSoftRst- PME-Enable- DSel=0 DScale=1 PME-
Capabilities: [d0] MSI: Enable+ Count=1/1 Maskable- 64bit+
Address: 00000000fee0300c Data: 41b9
Capabilities: [e0] PCI Advanced Features
AFCap: TP+ FLR+
AFCtrl: FLR-
AFStatus: TP-
Kernel driver in use: e1000e
Kernel modules: e1000e
```

「Capabilities: [d0] MSI」の「Address」と「Data」の所ですが、これをビットフィールドと突き合わせて読まないといけません。分かりにくいですね。なので、lspci を改造してわかり易く表示出来るようにしてみます。こちらが改造後のコード^{*4}になります。早速実行してみます。

```
# gcc -lpci msireg.c
# ./a.out 00:19.0
Message Signalled Interrupts: 64bit+ Queue=0/0 Enable+
address_hi=0
address_lo=fee0300c dest_mode=logical redirection=lowpri dest_id=3
data=41b9 trigger=edge level=assert delivery_mode=lowpri vector=185
```

Logical mode で Lowpri、destid=3、vector=185 になってるのが分かります。ここで smp_affinity を変えてみましょう。

```
# echo 1 > /proc/irq/44/smp_affinity
# ./a.out 00:19.0
Message Signalled Interrupts: 64bit+ Queue=0/0 Enable+
address_hi=0
address_lo=fee0100c dest_mode=logical redirection=lowpri dest_id=1
data=41b9 trigger=edge level=assert delivery_mode=lowpri vector=185
```

dest_id が 1 に書き換わったのが見て取れます。

ライセンス

Copyright (c) 2014 Takuya ASADA. 全ての原稿データ はクリエイティブ・コモンズ 表示 - 継承 4.0 国際ライセンスの下に提供されています。

参考文献

“BIOS が PCI Express を初期化する手順が見えてきた: なひたふ JTAG 日記。” <http://nahitafu.cocolog-nifty.com/nahitafu/20>

^{*4} <https://gist.github.com/1568777>