Most (all) Intel-MP compliant SMP boards have the so-called 'IO-APIC', which is an enhanced interrupt controller. It enables us to route hardware interrupts to multiple CPUs, or to CPU groups. Without an IO-APIC, interrupts from hardware will be delivered only to the CPU which boots the operating system (usually CPU#0).

Linux supports all variants of compliant SMP boards, including ones with multiple IO-APICs. Multiple IO-APICs are used in high-end servers to distribute IRQ load further.

There are (a few) known breakages in certain older boards, such bugs are usually worked around by the kernel. If your MP-compliant SMP board does not boot Linux, then consult the linux-smp mailing list archives first.

If your box boots fine with enabled IO-APIC IRQs, then your /proc/interrupts will look like this one:

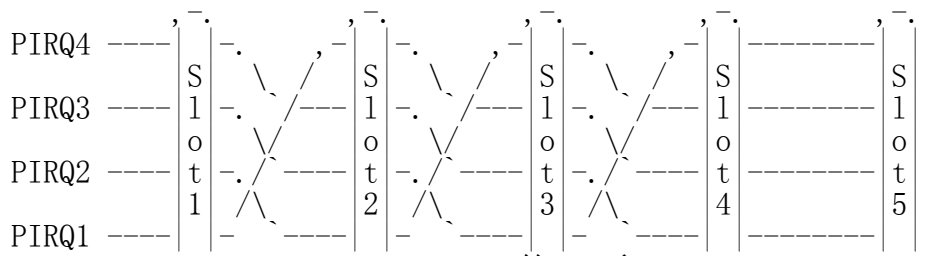
```
  ---------------------------->
 hell:~> cat /proc/interrupts
           CPU0
  0:    1360293    IO-APIC-edge  timer
  1:          4    IO-APIC-edge  keyboard
  2:          0          XT-PIC  cascade
 13:          1          XT-PIC  fpu
 14:       1448    IO-APIC-edge  ide0
 16:      28232   IO-APIC-level  Intel EtherExpress Pro 10/100 Ethernet
 17:      51304   IO-APIC-level  eth0
NMI:          0
ERR:          0
 hell:~>
 <----------------------------
```

Some interrupts are still listed as 'XT PIC', but this is not a problem; none of those IRQ sources is performance-critical.

In the unlikely case that your board does not create a working mp-table, you can use the pirq= boot parameter to 'hand-construct' IRQ entries. This is non-trivial though and cannot be automated. One sample /etc/lilo.conf entry:

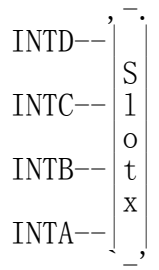
```
append="pirq=15,11,10"
```

The actual numbers depend on your system, on your PCI cards and on their PCI slot position. Usually PCI slots are 'daisy chained' before they are connected to the PCI chipset IRQ routing facility (the incoming PIRQ1-4 lines):



`_'  `_' `_'  `_' `_'

Every PCI card emits a PCI IRQ, which can be INTA, INTB, INTC or INTD:



These INTA-D PCI IRQs are always 'local to the card', their real meaning depends on which slot they are in. If you look at the daisy chaining diagram, a card in slot4, issuing INTA IRQ, it will end up as a signal on PIRQ4 of the PCI chipset. Most cards issue INTA, this creates optimal distribution between the PIRQ lines. (distributing IRQ sources properly is not a necessity, PCI IRQs can be shared at will, but it's a good for performance to have non shared interrupts). Slot5 should be used for videocards, they do not use interrupts normally, thus they are not daisy chained either.

so if you have your SCSI card (IRQ11) in Slot1, Tulip card (IRQ9) in Slot2, then you'll have to specify this pirq= line:

```
append="pirq=11,9"
```

the following script tries to figure out such a default pirq= line from your PCI configuration:

```
echo -n pirq=; echo `scanpci | grep T_L | cut -c56-` | sed 's/ /,/g'
```

note that this script wont work if you have skipped a few slots or if your board does not do default daisy-chaining. (or the IO-APIC has the PIRQ pins connected in some strange way). E.g. if in the above case you have your SCSI card (IRQ11) in Slot3, and have Slot1 empty:

```
append="pirq=0,9,11"
```

[value '0' is a generic 'placeholder', reserved for empty (or non-IRQ emitting) slots.]

Generally, it's always possible to find out the correct pirq= settings, just permute all IRQ numbers properly ... it will take some time though. An 'incorrect' pirq line will cause the booting process to hang, or a device won't function properly (e.g. if it's inserted as a module).

If you have 2 PCI buses, then you can use up to 8 pirq values, although such boards tend to have a good configuration.

Be prepared that it might happen that you need some strange pirq line:

```
append="pirq=0,0,0,0,0,0,9,11"
```

Use smart trial-and-error techniques to find out the correct pirq line ...

Good luck and mail to linux-smp@vger.kernel.org or
linux-kernel@vger.kernel.org if you have any problems that are not covered
by this document.

-- mingo