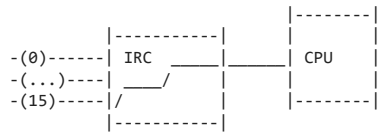
Interrupt handling (Linux/General)

On the mainbord we have an interrupt controller (IRC) which acts as a multiplexer between the devices which can raise an interrupt and the CPU:



Every device is associated with an IRQ (the number on the left). After every execution the CPU senses the interrupt-request line. If a signal is detected a state save will be performed and the CPU loads an Interrupt Handler Routine which can be found in the Interrupt Vector which is located on a fixed address in memory. As far as I can see the Number of the IRQ and the Vector number in the Interrupt Vector are not the same because I have for example my network card registered to IRQ 8. On an Intel Pentium processor this would point to a routine which is used to signal one error condition so there must be a mapping somewhere which points to the correct handler.

Questions:

1) If I write an device driver and register an IRQ X for it. From where does the system know which device should be handled? I can for example use request_irq() with IRQ number 10 but how does the system know that the handler should be used for the mouse or keyboard or for whatever i write the driver?

2) How is the Interrupt Vector looking then? I mean if I use the IRQ 10 for my device this would overwrite an standard handler which is for error handling in the table (the first usable one is 32 according to Silberschatz (Operating System Concepts)).

3) Who initialy sets the IRQs? The Bios? The OS?

4) Who is responsible for the matching of the IRQ and the offset in the Interrupt Vector?

5) It is possible to share IRQS. How is that possible? There are hardware lanes on the Mainboard which connect devices to the Interrupt Controller. How can to lanes be configured to the same Interrupt? There must be a table which says lane 2 and 3 handle IRQ15 e.g. Where does this table reside and how is it called?

linux operating-system interrupt

asked Jun 3 '12 at 10:27


fliX
135 2 13

A more detailed explanation on interrupt setup, handling and mapping can be found here: A code walk inside kernel interrupt framework – user2837221 Oct 2 '13 at 4:02

1 Answer

Answers with respect to linux kernel. Should work for most other OS's also.

1) If I write an device driver and register an IRQ X for it. From where does the system know which device should be handled? I can for example use request_irq() with IRQ number 10 but how does the system know that the handler should be used for the mouse or keyboard or for whatever i write the driver?

There is no 1 answer to it. For example if this is a custom embedded system, the hardware designer will tell the driver writer "I am going to route device x to irq y". For more flexibility, for example for a network card which generally uses PCI protocol. There are hardware/firmware level arbitration to assign an irq number to a new device when it is detected. This will then be written to one of the PCI configuration register. The driver first reads this device register and

then registers its interrupt handler for that particular irq. There will be similar mechanisms for other protocols.

What you can do is look up calls to request_irq in kernel code and how the driver obtained the irq value. It will be different for each kind of driver.

The answer to this question is thus, the system doesn't know. The hardware designer or the hardware protocols provide this information to driver writer. And then the driver writer registers the handler for that particular irq, telling the system what to do in case you see that irq.

2) How is the Interrupt Vector looking then? I mean if I use the IRQ 10 for my device this would overwrite an standard handler which is for error handling in the table (the first usable one is 32 according to Silberschatz (Operating System Concepts)).

Good question. There are two parts to it.

a) When you request_irq (irq,handler). The system really doesn't program entry 0 in the IVT or IDT. But entry N + irq. Where N is the number of error handlers or general purpose exceptions supported on that CPU. Details vary from system to system.

b) What happens if you erroneously request an irq which is used by another driver. You get an error and IDT is not programmed with your handler.

Note: IDT is interrupt descriptor table.

3) Who initialy sets the IRQs? The Bios? The OS?

Bios first and then OS. But there are certain OS's for example, MS-DOS which doesn't reprogram the IVT set up by BIOS. More sophisticated modern OS's like Windows or Linux do not want to rely on particular bios functions, and they re-program the IDT. But bios has to do it initially only then OS comes into picture.

4) Who is responsible for the matching of the IRQ and the offset in the Interrupt Vector?

I am really not clear what you mean. The flow is like this. First your device is assigned an irq number, and then you register an handler for it with that irq number. If you use wrong irq number, and then enable interrupt on your device, system will crash. Because the handler is registered fro wrong irq number.

5) It is possible to share IRQS. How is that possible? There are hardware lanes on the Mainboard which connect devices to the Interrupt Controller. How can to lanes be configured to the same Interrupt? There must be a table which says lane 2 and 3 handle IRQ15 e.g. Where does this table reside and how is it called?

This is a very good question. Extra table is not how it is solved in kernel. Rather for each shared irq, the handlers are kept in a linked list of function pointers. Kernel loops through all the handlers and invokes them one after another until one of the handler claims the interrupt as its own.

```
The code looks like this:

driver1:

d1_int_handler:
       if (device_interrupted()) <------------- This reads the hardware
       {
           do_interrupt_handling();
           return MY_INTERRUPT;
       }else {
           return NOT_MY_INTERRUPT;
       }

driver2:
       Similar to driver 1


kernel:
      do_irq(irq n)
      {
           if (shared_irq(n))
           {
                irq_chain = get_chain(n);
                while(irq_chain)
                {
                    if ((ret = irq_chain->handler()) == MY_INTERRUPT)
                        break;
                    irq_chain = irq_chain->next;
                }
                if (ret != MY_INTERRUPT)
```

```
                error "None of the drivers accepted the interrupt";
            }
        }
```

edited Jun 3 '12 at 18:23

answered Jun 3 '12 at 17:57

Saurabh

728 4 13

2 Excellent answer, thx! – fliX Jun 3 '12 at 18:20