Programming the PIC

by Alexander Blessing

What is the PIC?

The PIC is a controller in the PC hardware that handles hardware interrupts (IRQ0, IRQ1, etc.). When there wasn't a PIC, we would have to poll the hardware devices regularly, this would be time expensive. So instead of that, when a hardware interrupt occurs, the PIC gets the signal and sends it to the CPU. The CPU stops execution and processes the interrupt handler. We actually we have two PICs. The first one is for IRQ0-IRQ7, and the second for IRQ8-IRQ15. We must have a connection between PIC1 and PIC2.

The CPU only knows interrupts - it makes no difference between hardware or software interrupts. So it's the job of the programmer to map the hardware interrupts into interrupts that the CPU understands. In Real Mode, the hardware interrupts are mapped to interrupt 8-15 (first PIC) and 70-77 (second PIC). But that's a problem when we get into Protected Mode because interrupt 8-15 are reserved for exceptions. That would mean that when we would get an exception, the CPU would call a handler for the keyboard, for example. So we have to remap the PICs to use other interrupts...

Remapping the PICs

Remapping the PICs is quite easy. It is done in the initialization of the PICs. We must initialize the PICs *before* using them, and this is done by sending some ICW (Initialization Commands Words) to the PICs controller. The procedure is as follow:

- send ICW1 to PIC1 (20h) and PIC2 (A0h)
- send ICW2 to 21h for the first PIC and 0A1h for the second PIC
- send ICW3 to 21h for the first PIC and 0A1h for the second PIC
- finally, send ICW4 to 21h for the first PIC and 0A1h for the second PIC

ICW1 is used to tell the PICs if a ICW4 is following and if the PIC is working in a cascaded PIC environment (that means, if they communicate with each other).
ICW2 is the thing that is important for us. It tells the PICs where to map IRQ0 and IRQ8.
ICW3 is used for telling the PICs which IRQ to use for the communication between each other
ICW4 is used for telling that we are working in a 80x86 architecture and if the interruption is handled automatically or if it needs help from software.

May seem difficult, but it's not. If you want to understand it exactly, visit this page. But now, let's look at a code example. This code will initialize the PICs, remap them and disable all IRQs:

```
#define PIC1 0x20
#define PIC2 0xA0

#define ICW1 0x11
#define ICW4 0x01

/* init_pics()
 * init the PICs and remap them
 */
void init_pics(int pic1, int pic2)
{
        /* send ICW1 */
        outb(PIC1, ICW1);
        outb(PIC2, ICW1);

        /* send ICW2 */
        outb(PIC1 + 1, pic1);   /* remap */
        outb(PIC2 + 1, pic2);   /*  pics */

        /* send ICW3 */
        outb(PIC1 + 1, 4);      /* IRQ2 -> connection to slave */
        outb(PIC2 + 1, 2);

        /* send ICW4 */
        outb(PIC1 + 1, ICW4);
        outb(PIC2 + 1, ICW4);

        /* disable all IRQs */
        outb(PIC1 + 1, 0xFF);
}
```

So, this call will remap the PICs so that IRQ0 starts at 0x20 and IRQ8 starts at 0x28:

```
init_pics(0x20, 0x28);
```

Summary

That's it - I hope you understood the basic things, and take a look at the link I just gave you above. You will need some knowledge of programming the PICs as a OS developer, so do not think it's not important.

best regards,
Alexander Blessing