Linux Device Driver

(Interrupt Handling)

Amir Hossein Payberah

payberah@yahoo.com

Contents

- introduction
- Installing an interrupt handler
- Implementing a handler
- Interrupt sharing

Introduction

- An interrupt is simply a signal that the hardware can send when it wants the processor’s attention.
- For the most part, a driver need only register a handler for its device’s interrupts, and handle them properly when they arrive.

Introduction

- There were just 16 interrupt lines and one processor to deal with them.
  - Modern hardware can have many more interrupts.
- Unix-like systems have used the functions cli and sti to disable and enable interrupts for many years.

Contents

- introduction
- Installing an interrupt handler
- Implementing a handler
- Interrupt sharing

Installing interrupt handler

- Interrupt lines are a precious and often limited resource.
- The kernel keeps a registry of interrupt lines (similar to the registry of I/O ports).
- A module is expected to request an interrupt channel before using it, and to release it when it’s done.

Installing interrupt handler

-
  ```
  int request_irq(unsigned int irq,
                  void (*handler)(int, void *, struct pt_regs *),
          unsigned long flags,
                  const char *dev_name,
                  void *dev_id);
  ```

-
  ```
  void free_irq(unsigned int irq, void *dev_id);
  ```
- They are defined in <linux/sched.h>.

Request_irq

- irq
  - This is the interrupt number being requested.
- void (*handler)(int, void *, struct pt_regs *)
  - The pointer to the handling function being installed.
- Flags
  - SA_INTERRUPT
  - SA_SHIRQ
- dev_name
  - The string passed to request_irq is used in /proc/interrupts to show the owner of the interrupt.
- void *dev_id
  - This pointer is used for shared interrupt lines. It is a unique identifier.

Installing place

- The correct place to call request_irq is when the device is first opened, before the hardware is instructed to generate interrupts.
- The place to call free_irq is the last time the device is closed, after the hardware is told not to interrupt the processor any more.

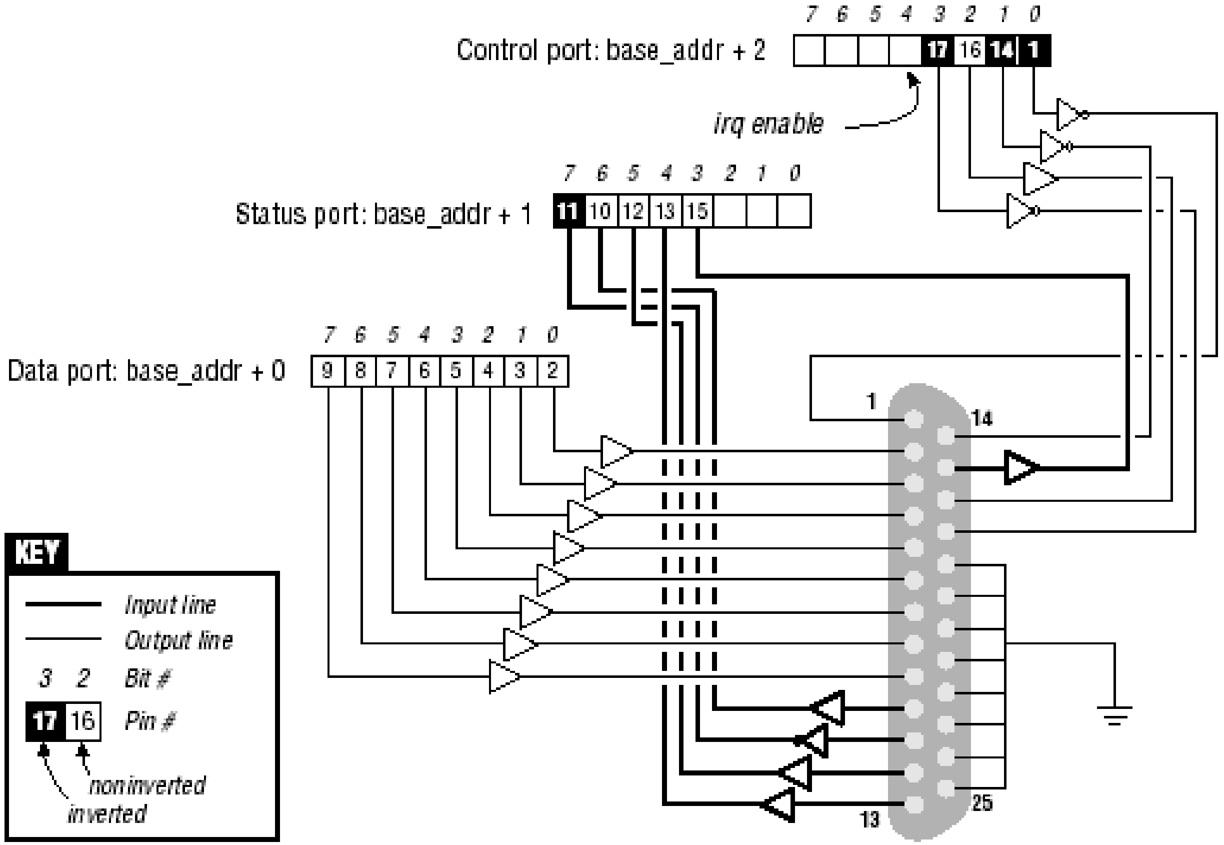
Auto detecting IRQ number

- One of the most compelling problems for a driver at initialization time can be how to determine which IRQ line is going to be used by the device.
- The Linux kernel offers a low-level facility for probing the interrupt number.
- It only works for nonshared interrupts.

Kernel-assisted probing

- unsigned long probe_irq_on(void);
  - This function returns a bit mask of unassigned interrupts.
  - The driver must preserve the returned bit mask and pass it to pr obe_irq_off later.
- int probe_irq_off(unsigned long);
  - After the device has requested an interrupt, the driver calls this function, passing as argument the bit mask previously returned by probe_irq_on.
  - probe_irq_off returns the number of the interrupt that was issued after ‘‘probe_on.’’
  - If no interrupts occurred, 0 is returned.
  - If more than one interrupt occurred probe_irq_off returns a negative value.
- They are defined in <linux/interrupt.h>.

Parallel port registers



Control port: base_addr + 2

7	6	5	4	3	2	1	0
				17	16	14	1

Status port: base_addr + 1

7	6	5	4	3	2	1	0
11	10	12	13	15			

Data port: base_addr + 0

7	6	5	4	3	2	1	0
9	8	7	6	5	4	3	2

Sample

```
unsigned long mask;
mask = probe_irq_on();
outb_p(0x10,short_base+2); /* enable reporting */
outb_p(0x00,short_base);     /* clear the bit */
outb_p(0xFF,short_base);     /* set the bit: interrupt! */
outb_p(0x00,short_base+2); /* disable reporting */
udelay(5);                             /* give it some time */
short_irq = probe_irq_off(mask);
if (short_irq == 0)
{
    printk(KERN_INFO "short: no irq reported by probe\n");
    short_irq = -1;
}
if (short_irq < 0)
    printk("short: probe failed %i times, giving up\n", count);
```

Do-it-yourself probing

```
int trials[] = {3, 5, 7, 9, 0},  tried[] = {0, 0, 0, 0, 0}, i;
for (i=0; trials[i]; i++)
      tried[i] = request_irq(trials[i], short_probing, SA_INTERRUPT, "short probe", NULL);
short_irq = 0; /* none obtained yet */
outb_p(0x10,short_base+2); /* enable */
outb_p(0x00,short_base);
outb_p(0xFF,short_base); /* toggle the bit */
outb_p(0x00,short_base+2); /* disable */
udelay(5); /* give it some time */
if (short_irq == 0)
{ /* none of them? */
      printk(KERN_INFO "short: no irq reported by probe\n");
}
for (i=0; trials[i]; i++)
      if (tried[i] == 0)
      free_irq(trials[i], NULL);
if (short_irq < 0)
printk("short: probe failed %i times, giving up\n", count);
```

Contents

- introduction
- Installing an interrupt handler
- Implementing a handler
- Interrupt sharing

Implementing a handler

- The role of an interrupt handler is to give feedback to its device about interrupt reception.
- And to read or write data according to the meaning of the interrupt being serviced.
- A typical task for an interrupt handler is awakening processes sleeping on the device.

Interrupt handler

- void (*handler)(int irq, void *dev_id, struct pt_regs *regs);

Sample

```
void irq_handle (int irq, void* dev, struct pt_regs*
    regs)
{
    wake_up_interruptible (&q);
}
//-----------------------------------------------------------------------
static int device_open (struct inode *inode, struct
    file *file)
{
    irq = request_irq (7, irq_handle, SA_INTERRUPT,
    “my_irq", NULL);
    return 0;
}
```

Contents

- introduction
- Installing an interrupt handler
- Implementing a handler
- ⇨ Interrupt sharing

Interrupt sharing

- In general, IRQ lines on the PC have not been able to serve more than one device,

Installing a Shared Handler

- Shared interrupts are installed through request_irq just like nonshared ones, but
- there are two differences:
  - □ The SA_SHIRQ bit must be specified
  - □ The dev_id argument must be unique.

Question?