Real Time and Embedded Systems

by

Dr. Lesley Shannon

Email: lshannon@ensc.sfu.ca

Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc351

Simon Fraser University

Slide Set: 8
Date: November 15, 2011

Slide Set Overview

- More on Interrupts in Linux

Check out chapter 10 of the Linux Device Driver book for even more details

Linux Interrupts

- Linux handles hardware interrupts similar to signals in user space.

- Generally, a driver just registers a handler for its device's interrupts that will handle them properly when they occur.

- However, interrupt handlers are rather limited in the actions they can perform – this effects how they run.

Parallel Port

- I'll be going through the discussion of the parallel port example from the book
  - Also commonly called a printer port

- IBM PC systems used to allocate their first three parallel ports according to the following table (from Wikipedia):

PORT NAME	Interrupt #	Starting I/O	Ending I/O
LPT1	IRQ 7	0x378	0x37f
LPT2	IRQ 5	0x278	0x27f
LPT3	IRQ 2	0x3bc	0x3bf

Parallel Port

- The traditional commandline for unix/linux to print is: *lpr* (you can cheque the print queue with *lpq*)
  - By default, you print to LPT1
- If an LPTx slot is unused, the port addresses of the other LPT ports may be moved up.
- However, the IRQ lines remain fixed (they are fabbed into the PCB board)

PORT NAME	Interrupt #	Starting I/O	Ending I/O
LPT1	IRQ 7	0x378	0x37f
LPT2	IRQ 5	0x278	0x27f
LPT3	IRQ 2	0x3bc	0x3bf

Linux Interrupts

- The printer driver is known as the “lp” driver (lpr, lpq, …)
- The parallel port uses an interrupt to inform the lp driver that it is ready to accept the next character in the buffer to print
- Remember, the hardware system has to be *configured* to generate interrupts before it will happen
- The parallel standard states that setting bit 4 of port 2 (0x37a/0x27a/Base Address+2…) enables interrupt reporting.
  - You can use *outb to set the bit*

Linux Interrupts

- With interrupts enabled on the device, the parallel port will generate an interrupt on Pin 10 (called its ACK bit)
- It is rising edge activated
- However, just because the device generates interrupts, doesn’t mean they are handled the way you want.
- By default, linux will simply acknowledge the interrupt and ignore it.

Linux Interrupts

- With interrupts enabled on the device, the parallel port will generate an interrupt on Pin 10 (called its ACK bit)

- It is rising edge activated

- However, just because the device generates interrupts, doesn’t mean they are handled the way you want.

- By default, linux will simply acknowledge the interrupt and ignore it.

Linux Interrupts

- Therefore, you also need to configure a software “handler” to service the interrupt

- Remember, there are only so many interrupt pins on the CPU :
  - If a device doesn’t need interrupts, don’t waste them

- The kernel keeps a registry of interrupt lines
  - It’s like the I/O registry
  - Remember your Interrupt Vector Table

Linux Interrupts

- The device has to request an interrupt channel (i.e. IRQ) before using it and is expected to release it when done.

- In many cases, a driver may have to share an interrupt line with other drivers
  - recall daisy chaining

- Checkout the functions in <linux/interrupt.h>

Linux Interrupts

- Checkout the functions in <linux/interrupt.h>:

```
int request_irq(unsigned int irq, irqreturn_t (*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);
```

Linux Interrupts

- Checkout the functions in <linux/interrupt.h>:

```
void free_irq(unsigned int irq, void *dev_id);
```

- This is the easy function with simple parameters, so we’re going to focus on request_irq

Linux Interrupts

- Checkout the functions in <linux/interrupt.h>:

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

- *request_irq* returns 0 to indicate success or a negative error code, as usual.
  - For example, it may return -EBUSY to indicate that another device driver is currently using the requested interrupt lline

Linux Interrupts

- request_irq’s arguments are:
  - unsigned int irq,
  - irqreturn_t (*handler)(int, void *, struct pt_regs *),
  - unsigned long flags,
  - const char *dev_name,
  - void *dev_id);

Linux Interrupts

- The flag bits that can be set are:
  - SA_INTERRUPT
  - SA_SHIRQ
  - SA_SAMPLE_RANDOM

Questions?

- How do you request an interrupt channel in linux?

- What function frees an interrupt channel?

Questions?

- Why may you need to free an interrupt channel?

- Be ready for “fast and slow handlers” and “interrupt sharing”…