Linux *Kernel Hacking* Free Course 2003

UNIVERSITA' DEGLI STUDI DI ROMA TorVergata Università

D. P. Bovet, M. Cesati, V. Garofalo

The lp_sensor device

Before moving on

- Before introducing a new circuit for the parallel port, let us ask whether we could have taken advantage of drivers already included in Linux
- In order to answer this question, we'll have a quick look at the Linux drivers for the serial and parallel ports

Linux drivers for the serial port

- Each serial port has a corresponding `/dev/ttyS0`, `/dev/ttyS1`, etc, device file
- Many different devices (mouse, serial printer, modem, etc.) can be connected to a serial port
- Linux includes a driver that allows to read from or write into a serial port
- Each device connected to a serial port is managed by a different user-space program

Linux drivers for the parallel port

- Parallel ports have device files `/dev/parport0`, `/dev/parport1`, ...
- If you select the ``Parallel port support'' option and the ``PC-style hardware'' suboption in menuconfig, you'll get the low-level driver for the parallel ports
- Allows user-space applications to interact directly with parport devices

Devices connected to parallel ports

- Most external devices connected to a parallel port have their own device files
- For instance, `/dev/lp0`, `/dev/lp1` for the parallel printers
- The corresponding device drivers (for instance, `linux/drivers/char/lp.c`) must use the low-level parport device driver
- You may see all device drivers using the low-level parport device driver by looking into the `/proc/sys/dev/parport` directory

Suggested reading

- `Documentation/parport.txt`
- `Documentation/parport-lowlevel.txt`
- `Documentation/paride.txt`
- The code:
  - `drivers/char/lp.c`
  - `drivers/parport/parport_pc.c`
  - `include/linux/parport.h`
  - `drivers/block/paride/paride.c`

Why are we making new drivers?

- Neither the lp_led device described in the previous lecture nor the lp_sensor one presented today use the low-level parport device driver
- Both circuits use the status lines of the parallel port in a non standard way
- New I/O drivers are thus necessary to support these circuits

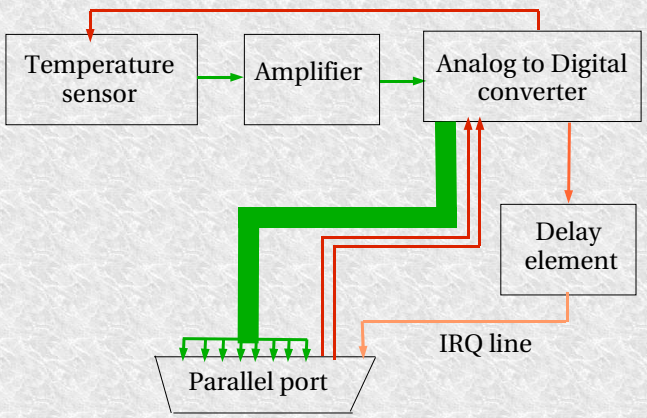
The lp_sensor device

- This device is connected to the parallel port
- It makes use of a temperature sensor
- Its response time is non negligible (this is done on purpose)
- The use of an interrupt to signal the end of a temperature reading is thus justified

The approach

- Describe the circuit
- Show how to program the circuit
- Describe the I/O driver
- Develop a simple user-space application to test the I/O driver

The circuit diagram



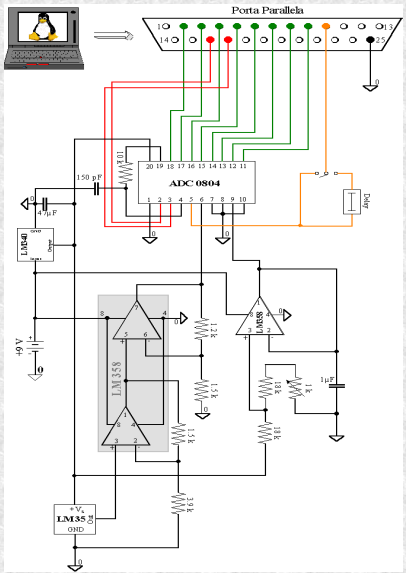
The LM35 temperature sensor

- Provides typical accuracies of 0.5 C° at room temperature
- Outputs 10 mV / C°
- Thermal response in still air requires a couple of minutes
- Must be triggered by a 4-20V signal

The ADC804 converter

- 8-bit A/D converter
- 0V to 5V analog input voltage
- 100 μs conversion time
- Pins 1-3 (all 3 negated) correspond, respectively, to CS, RD, and WR functions
- Pin 5 (negated) corresponds to the interrupt line

The circuit layout



The Data port

Pins: P_9 P_8 P_7 P_6 P_5 P_4 P_3 P_2

Bits: D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

- The 8 lines of the data port are connected to pins 11-18 of the A/D converter

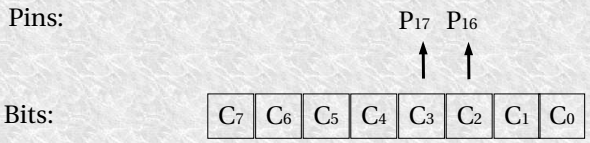
The Status port

Pins: P_{10}

Bits: S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0

- Bit 6 of the status port is connected via pin 10 to pin 5 of the A/D converter

The Control port



- Bit C4 is used by the interface as an Interrupt Enable Flag
- Bit C3 is connected, via pin 16, to pin 2 (negated RD) of the A/D converter
- Bit C2 is connected, via pin 17, to pin 3 (negated Write) of the A/D converter

An I/O driver for the lp_sensor device

Main I/O driver characteristics

- Closely inspired to the one included in the `linux/drivers/char/lp.c` file
- Supports the `open()`, `close()`, and `read()` file operations
- Can be configured as a static kernel component or as a module

Steps to be performed (1)

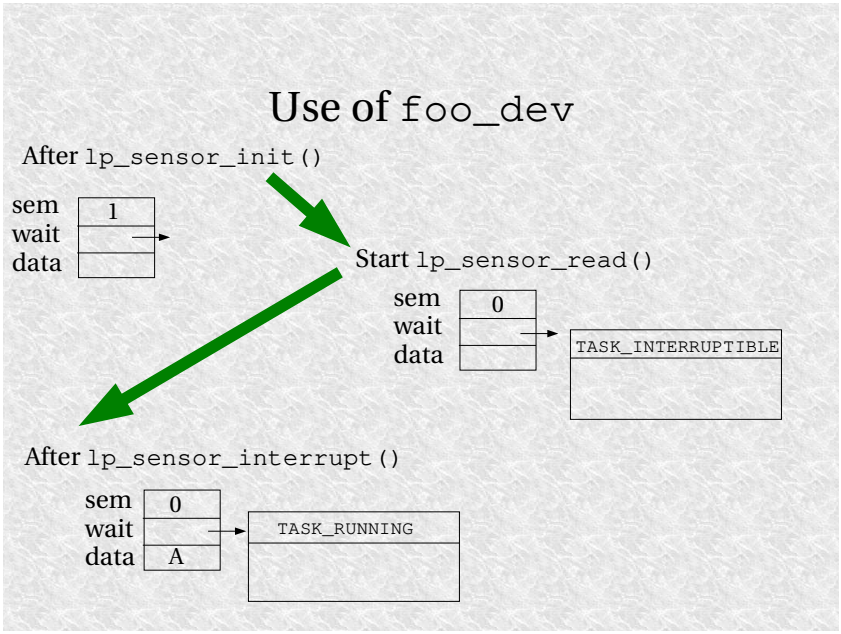
- Add a new kernel configuration option
- Select a free major number (minor=0)
- Create a new `/dev/lp_sensor` device file for the lp_sensor device
- Insert a new initialization function in `chr_dev_init()` (to be invoked only if the driver is compiled statically)

Steps to be performed (2)

- Add the functions that the file operations of the new device
- Write a suitable test program
- Compile the new kernel and boot it
- The details are shown in the `sensor_device.txt` file

Handling the interrupt

- A critical data structure called `foo_dev` includes the following main fields:
  - A `wait` list head data structure where the process will wait waiting for the interrupt to occur
  - An 8-bit field `data` where the data read from the device will be stored
  - A `sem` semaphore to serialize accesses
- `foo_dev` interfaces `lp_sensor_read()` with `lp_sensor_interrupt()`


Use of foo_dev
After lp_sensor_init()
sem
wait
data
1
Start lp_sensor_read()
sem
wait
data
0
TASK_INTERRUPTIBLE
After lp_sensor_interrupt()
sem
wait
data
0
A
TASK_RUNNING