Computer Labs: The PC Keyboard
Lab 3: Part 2

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

October 26, 2014

Lab 3: `kbd_test_scan(ushort asm)`

What
: Prints the scancodes as in part 1, i.e. both the **makecode** and the **breakcode**, read from the KBC

- Should terminate when it reads the **breakcode** of the `ESC` key: `0x81`
- The first byte of two byte scancodes is usualy `0xE0`
  - This applies to both make and break codes

How
: If `asm` is non-zero, call interrupt handler (IH) written in **assembly**. Upon an interrupt:

- read the scancode from the `OUT_BUF`
- put the scancode in some variable

IMP

1. Must use **linked** assembly
   - May use Intel syntax, if supported
   - Use **.S** suffix, if you use CPP's directives (e.g. `#include`)
2. The variable must be defined in **assembly**

Do not forget
: Minix already has an IH installed

Minix 3 Notes: I/O In Assembly

Problem How can assembly code execute I/O operations?

- Minix 3 device drivers, and your programs, execute at user-level.

Solution Two possible solutions:

1. Use `sys_inX()/sys_outX()` kernel calls
   - That is, make the kernel calls from assembly
2. Use the I/O privilege field in the EFLAGS register, via the `sys_iopenable()` kernel call

Minix 3 Notes: `sys_iopenable()` (1/2)

`sys_iopenable()`
: "Enable the CPU's I/O privilege level bits for the given process, so that it is allowed to directly perform I/O in user space."

I/O privilege level (IOPL) field (2 bits) in the EFLAGS register

- Specifies the privilege level of a process, so that it can perform the following operations
  - `IN/OUT`
  - `CLI` (disable interrupts)
  - `STI` (enable interrupts)

Minix 3 Notes: `sys_iopenable()` (2/2)

Note `sys_iopenable()` is a blunt mechanism

- The process is granted the permission to perform I/O on any I/O port
  - Need to grant permission in `/etc/system.conf.d/XXXX`
- With `sys_inX()/sys_outX()` the I/O operations are executed by the (micro)kernel and it is possible to grant permission to only a few selected I/O ports (as determined by `/etc/system.conf.d/XXXX`)

Lab 3: `kbd_test_timed_scan(ushort idle)`

What Similar to `kbd_test_scan()` except that process should terminate, upon:

either release of the ESC key

or after `idle` seconds, during which no scancode is received

How Must subscribe interrupts both of the keyboard and the timer/counter

- Must handle both interrupts in the "`driver_receive()` loop"

```
        switch (_ENDPOINT_P(msg.m_source)) {
        case HARDWARE: /* hardware interrupt notification */
            if (msg.NOTIFY_ARG & irq1) { /* subscribed interrupt */
                ... /* process it */
            }
            if (msg.NOTIFY_ARG & irq2) { /* subscribed interrupt */
                ... /* process it */
            }
            break;
        default:
            break; /* no other notifications expected: do nothing */
        }
```

- Must not change timer 0's configuration