Computer Labs: The PC Keyboard
Lab 3: Part 2

2º MIEIC

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

October 15, 2018

Lab 3: `kbd_test_scan(bool assembly)`

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`
- No need to print the number of calls of `sys_inb()`

How If `assembly` is true, call interrupt handler (IH) written in **assembly**.

IMP

1. Must use **linked** assembly
   - May use Intel's syntax, if you wish
   - Use **.S** suffix, if you use CPP's directives (e.g. `#include`)
2. Must use a variable defined in **assembly** to pass the byte read from the output buffer by the IH to the remaining of the code in C

Do not forget Minix already has an IH installed

Lab 3: Assembly Interrupt Handler

- The IH is all the code that interacts with the KBC upon occurrence of a keyboard interrupt (must be reusable):
  1. Read the status register
     - For checking communication errors
  2. Read the output buffer
  - Must put the byte read in a variable defined in assembly
- Everything else, should be written in C. E.g.:
  - KBD interrupts subscription/unsubscription;
  - Interrupt loop;
  - Printing of the scancode

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`)

IMP You need not call `sys_iopenable()`, `lcf_start()` calls it before calling `kbd_test_scan()`, with the argument set to true

Lab 3: Configuration File

```
 service lab3
{
        system
                DEVIO
                IRQCTL
                IOPENABLE
                ;
        ipc
                SYSTEM
               [...]
                vfs
                ;
        io
                40:4
                60
                64
                ;
        irq
                0          # TIMER 0 IRQ
                1          # KBD IRQ
                ;
        uid     0
                ;
};
```

Lab 3: `kbd_test_timed_scan(uint8_t 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"

```
12:         switch (_ENDPOINT_P(msg.m_source)) {
13:         case HARDWARE: /* hardware interrupt notification */
14:             if (msg.NOTIFY_ARG & irq_timer0) { /* subscribed interrupt
15:                 ... /* process timer0 interrupt request     */
16:             }
17:             if (msg.NOTIFY_ARG & irq_kbd) { /* subscribed interrupt */
18:                 ... /* process KBD interrupt request  */
19:             }
20:             break;
21:         default:
22:             break; /* no other notifications expected: do nothing */
23:         }
```

- Must not change timer 0's configuration