Overview

- Xenomai #3
- Xenomai #6, #7, #8
- Interrupts
- Explanation Xenomai #9 (parallel port)
- Priority inversion at NXP
- Lab visit (or at 13:45): room 0.12 Mercator 1

Key (in this order):
- Harco Kuppens: room 0.09
- Simone Meeuwsen: room 0.06
- Bernadette Smelik: room 0.11

**Instruction** @ 13:45:
- individual help in lab - room 0.12 Mercator 1

Interrupts

- Avoid that processor is kept busy with polling loops for asynchronous external events.

Programmable Interrupt Controller (PIC)

- Manages hardware interrupts:
  - priorities
  - masking
  - acknowledgements (indicating CPU received interrupt)

Older PC has 15 HW IRQ lines

<table>
<thead>
<tr>
<th>Bit</th>
<th>Disable</th>
<th>IRQ</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>IRQ7</td>
<td>Parallel Port</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>IRQ6</td>
<td>Floppy Disk Controller</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>IRQ5</td>
<td>Reserved/Sound Card</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IRQ4</td>
<td>Serial Port</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>IRQ3</td>
<td>Serial Port</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>IRQ2</td>
<td>PIC2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>IRQ1</td>
<td>Keyboard</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>IRQ0</td>
<td>System Timer</td>
<td></td>
</tr>
</tbody>
</table>

ISR should be fast and non-blocking (e.g. avoid semaphores)
### PIC2

<table>
<thead>
<tr>
<th>Bit</th>
<th>IRQ</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>IRQ15</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>IRQ14</td>
<td>Hard Disk Drive</td>
</tr>
<tr>
<td>5</td>
<td>IRQ13</td>
<td>Maths Co-Processor</td>
</tr>
<tr>
<td>4</td>
<td>IRQ12</td>
<td>PS/2 Mouse</td>
</tr>
<tr>
<td>3</td>
<td>IRQ11</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>IRQ10</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>IRQ9</td>
<td>Redirected IRQ2</td>
</tr>
<tr>
<td>0</td>
<td>IRQ8</td>
<td>Real Time Clock</td>
</tr>
</tbody>
</table>

Newer PCs might have Advanced PIC (APIC); allows up to 255 physical hardware IRQ lines, typically \( \approx 24 \) supported.

### Types of Interrupts

- **Level-triggered interrupt:** indicated by high level (1) or low level (0) of IRQ line. Device drives IRQ line to its active level and holds it at that level until serviced.
- **Edge-triggered interrupt:** signaled by a level transition on the interrupt line, either a falling edge (1 to 0) or a rising edge (0 to 1).
- **Hybrid of level-triggered and edge-triggered signaling:** hardware not only looks for an edge, but it also verifies that the interrupt signal stays active for a certain period of time. Common use: for non-maskable interrupt input, because they often signal major – or even catastrophic – system events.

### Interrupts

- **Maskable interrupt:** hardware interrupt that may be ignored by setting bit in interrupt mask register’s (IMR) bit-mask.
- **Non-maskable interrupt** (NMI): hardware interrupt without associated bit-mask - it can never be ignored. Often used for timers, especially watchdog timers.
- **Interprocessor interrupt:** generated by one processor to interrupt another processor in a multiprocessor system.
- **Software interrupt:** generated within processor by executing instruction; interrupt 0x80 often used to execute system calls by user mode program.
- **Spurious interrupt:** unwanted hardware interrupt. E.g., by electrical interference on an interrupt line or through incorrectly designed hardware.

### Registers of Programmable Interrupt Controller (PIC):

- **Interrupt Request Register (IRR):** specifies which interrupts are pending acknowledgement.
- **In-Service Register (ISR):** specifies which interrupts have been acknowledged, but are still waiting for an End Of Interrupt (EOI).
- **Interrupt Mask Register (IMR):** specifies which interrupts are to be ignored and not acknowledged. Allows up to two distinct interrupt requests to be outstanding at one time, one waiting for acknowledgement, and one waiting for EOI.

**End Of Interrupt (EOI)** is sent to PIC to indicate completion of interrupt processing for given interrupt. Then more interrupt requests of same type can be generated by the PIC.

### Interrupts in Xenomai via Adeos pipeline

### Adeos pipeline
Interrupts in Xenomai kernel

Xenomai allows interrupts handler as user space task

Exercise 9a: count keyboard presses

We work in user space with interrupts. Declare:

```c
#define KEYBOARD_IRQ 1
```

In main: (see [http://www.cs.ru.nl/lab/xenomai/api/](http://www.cs.ru.nl/lab/xenomai/api/))

- create interrupt
  ```c
  rt_intr_create(&keypress, NULL, KEYBOARD_IRQ, I_PROPAGATE)
  ```
- create and start handler task (isr)
  ```c
  rt_task_create(&key_isr, NULL,0,50,0)
  rt_task_start(&key_isr, &key_handler, NULL)
  ```

Define isr task:

```c
void key_handler(void *arg) {
  int nr_interrupts;
  ...
  while(1) {
      nr_interrupts = rt_intr_wait(&myinterrupt,TM_INFINITE);
      if (nr_interrupts_waiting>0) {
          .....
      }
  }
}
```

Note: pause() at the end of main

Exercise 9b

9b: [on real linux PC only – no VMware / VirtualBox]

count interrupts generated on parallel port

Parallel port

Parallel port consists of 3 bytes:

- Data port: 8 bits D0 … D7
- Status port: 5 in (read only) (S3 … S7), 3 unused
- Control port: 4 out (r/w) (C0 … C3), 4 setup/unused

Note: not all bits corresponds to physical pins. E.g., C4 enables interrupts
Generate interrupt
If interrupts enabled, i.e., bit C4 has been set and voltage on S6 (pin 10) is raised from 0 V to +5 V (e.g., by connecting it to ground [one of pins 19-25] and then disconnecting it, as done by push button) then parallel port generates interrupt on IRQ 7.

Enable/disable interrupts
- Address of control port byte: 0x37A
- Set C4 to enable interrupts before starting ISR task
  ```c
  ioperm(0x37A, 1, 1);    /* set port input/output permissions to read 1 byte*/
  byte = inb(0x37A);       /* get the byte */
  byte = byte | 0x10;       /* hex 10 = binary 00010000 */
  outb(byte, 0x37A);       /* output the byte */
  ```
- Reset C4 to disable interrupts at the end of main
  ```c
  byte = inb(0x37A);
  byte = byte & 0xEF;      /* hex EF = binary 11101111 */
  outb(byte, 0x37A);
  ```

See, for instance, [http://tldp.org/HOWTO/IO-Port-Programming-2.html](http://tldp.org/HOWTO/IO-Port-Programming-2.html)

NOTE: on lab PCs result may depend on setting of previous programs.

Exercise 9

- Needed:
  ```c
  #include <stdio.h>
  #include <signal.h>
  #include <unistd.h>
  #include <sys/mman.h>
  #include <native/intr.h>    // for interrupts
  #include <sys/io.h>        // for 9b: inb, outb
  ```
- print prio with
  ```c
  curtask = rt_task_self();
  rt_task_inquire(curtask,&curtaskinfo);
  rt_printf("Prio: \%d \n", curtaskinfo.cprio);
  ```

Linux PCs in the lab

root, pwd: ...

File transfer:
- connect to internet via cable (disable wireless)
  (you should have IP address 131.174.12.129)
- start Linux PC, login, execute: `ifconfig`
- check inet addr: 131.174.12.229 (for instance)
- On Windows in Explorer: `\131.174.12.229\root`
- login with root
  Alternative: SSH

  ➔ remove your files when ready !!!!!

USB on Linux PCs in the lab

- USB:
  ```
  – see how mounted, e.g. "ls /dev/sd*"
  – mkdir /mnt/usbstick
  – mount /dev/sda1 /mnt/usbstick
  – work with files in /mnt/usbstick
  – to stop: umount /mnt/usbstick
  ```
  ➔ remove your files when ready !!!!!

Assignment for 1 October 2014

- Xenomai exercise #9

Mail me before 30 September 18:00:
- Source files + Makefile + short explanation

NOTE: #9b can only be tested in the lab ➔ don’t wait till the last moment