Industrial hardware and QEMU

LinuxCon Europe 2012, Barcelona
Alberto Garcia <agarcia@igalia.com>
Introduction
Who am I?

- Alberto Garcia
  - Computer engineer, Coruña University
  - Working at Igalia since 2001
  - Experience in operating systems
  - Debian maintainer
  - Involved in Maemo and MeeGo
How to make the development of device drivers easier:

- Debug problems that are hard to reproduce
- Foresee points of failure
- Industrial hardware as an example
- Useful for any kind of hardware
Industrial hardware
Industrial hardware

- Used by:
  - Companies: factories, shipyards, robotics, ...
  - Research laboratories
  - Universities
Typical features

- Simple
- Reliable
- Well-tested technology
- Expensive / hard to find
- Hard to replace
- Sometimes designed ad-hoc
Types of industrial hardware

- ADCs, DACs
- Waveform Generators
- Time to digital converters
- Communications: serial ports, fieldbus
What it is used for

- Actuators
- Sensors
- Control systems
Examples

Struck SIS3320-250 8 channel ADC
Challenges
Challenges of developing for this hardware

- Limited availability
- Bugs in the production environment cannot be reproduced in the laboratory
- Difficult to debug on-site
  - Narrow time windows
  - Dangerous environment
Challenges of developing for this hardware
Debugging

- Is it a problem in the driver or in the device?
- Is the firmware faulty? Is it wrongly loaded?
- Is the hardware damaged?
- How can we reproduce the bug?
- Do we have easy access to the environment?
- Is it remotely located?
Using virtualization to make things easier
- Machine emulator and virtualizer
- Supports multiple architectures
- Can be accelerated using KVM
- Free software
- Flexible
How QEMU can help us

- We can emulate the device we’re working with
- Even if the hardware doesn’t exist yet!
- Useful not just for software:
  - Develop hardware and its driver in parallel
- Simulate hard to reproduce conditions
  - Force hardware errors or other uncommon situations
  - Foresee potential points of failure
  - Make the driver more reliable
Hardware we’ve worked with

- Simple PCIe FMC Carrier (SPEC)
- FMC Time to Digital Converter
How to create new devices in QEMU

```c
static void spec_class_init(ObjectClass *klass, void *data)
{
    DeviceClass *dc = DEVICE_CLASS(klass);
    PCIDeviceClass *k = PCI_DEVICE_CLASS(klass);

    k->is_express = true;
    k->init = spec_initfn;
    k->exit = spec_exitfn;
    k->vendor_id = PCI_VENDOR_ID_CERN;
    k->device_id = PCI DEVICE_ID_CERN SPEC;
    k->subsystem_vendor_id = PCI_VENDOR_ID_GENNUM;
    k->subsystem_id = PCI DEVICE_ID GENNUM GN4124;
    k->revision = 0x03;

    dc->desc = "Simple PCIe FMC carrier (SPEC)";
    dc->props = spec_properties;
    dc->vmstate = &vmstate_spec;
}

static const TypeInfo spec_info = {
    .name = "spec",
    .parent = TYPE_PCI_DEVICE,
    .instance_size = sizeof(SPECState),
    .class_init = spec_class_init,
};

static void spec_register_types(void)
{
    type_register_static(&spec_info);
}
```
Emulating the device

```c
static uint64_t tpci200_read_las0(void *opaque, target_phys_addr_t addr,
                                  unsigned size)
{
    TPCI200State *s = opaque;
    uint64_t ret = 0;

    switch (addr) {
    case REG_REV_ID:
        DPRINTF("Read REVISION ID\n"); /* Current value is 0x00 */
        break;

    case REG_IP_A_CTRL:
    case REG_IP_B_CTRL:
    case REG_IP_C_CTRL:
    case REG_IP_D_CTRL:
        {
            unsigned ip_n = IP_N_FROM_REG(addr);
            ret = s->ctrl[ip_n];
            DPRINTF("Read IP \%c CONTROL: 0x%x\n", 'A' + ip_n, (unsigned) ret);
        }
        break;

    case REG_RESET:
        DPRINTF("Read RESET\n"); /* Not implemented */
        break;

    case REG_STATUS:
        ret = s->status;
        DPRINTF("Read STATUS: 0x%x\n", (unsigned) ret);
    }
```
Testing our drivers
Tests

- Create a set of tests for the device
- Tests can “talk” to QEMU and inject errors or force other conditions
- Detect problems before the changes are put into production
Test environment

- Framework written in python
- Simple process:
  - The virtual machine is started
  - The test environment is set up
  - Tests are run
  - The results are compared with the expected values
Unattended testing

- Run QEMU without user interaction
- No display necessary
- Select the kernel that you want to boot
- Log all messages, compare them against expected values
- Easily share files between both sides
- Detect when the system crashes
Running the tests

```
berto@localhost:~/tests $ ./qemu-test run spec-tdc-tests
1..8
ok 1 spec-tdc-tests/read-chan0: Test reading on channel 0
ok 2 spec-tdc-tests/read-chan1: Test reading on channel 1
ok 3 spec-tdc-tests/read-chan2: Test reading on channel 2
ok 4 spec-tdc-tests/read-chan3: Test reading on channel 3
ok 5 spec-tdc-tests/read-chan4: Test reading on channel 4
not ok 6 spec-tdc-tests/time-threshold: Test time threshold behaviour.
ok 7 spec-tdc-tests/read-all-chans: Test reading on all channels.
ok 8 spec-tdc-tests/read-disabled-chans: Test reading with disabled channels.

Summary: succeeded 7, skipped 0, failed 1 (total 8)

Failed test cases (1):
  - time-threshold
```

```
berto@localhost:~/tests $ 
```
Continuous integration

- We can run tests automatically when the driver is updated
- Also if the hardware specs change
- Detect regressions more easily
Conclusions
Overall benefits

- No need to have the actual hardware
- You can also help the development of the hardware
  - Hardware and software can be developed in parallel
- Distributed teams
- More stable drivers
- Test changes before they are deployed
  - Faster development
- Continuous integration
- Contribute to QEMU
Questions & Answers
Thank you!
Images used in this presentation

- http://www.flickr.com/photos/9197427@N06/2885827751
- http://www.flickr.com/photos/twosevenoneonenineeightthreesevenatenzerosix/7848902444/
- http://www.flickr.com/photos/11304375@N07/2046228644/