Testing QEMU emulated devices using qtest

Marc Marí Barceló <marc.mari.barcelo@gmail.com>

KVM Forum 2014

Who am I?

Computer Science student

Worked on QEMU in GSoC project

Other hacking activities:

- Satellite software
- Android Real Time Operating System

What will I talk about?

- Introduction
- What is a QTest? What is libqos?
- How are devices accessed?
- Basic test structure
- Libqos API functions
- Debugging and testing
- Conclusion

Index

Why are QTests necessary?

- QEMU emulates hardware
- Acceptance test: checks hardware works as expected.
- How to verify specification compliant?

Qtests: directly test emulated devices without running a full guest.

Who uses QTests?

- Developers:
  - Test cases for new devices
  - Regression tests for bugs
- Testers:
  - Automate tests
  - Exercise error paths (by broken or malicious guests)

Index

GLib tests

- GLib provides a unit testing framework
- QTests are based on GLib testing framework
- GLib provides:
  - Test cases: methods
  - Test suite: group of test cases

Source: https://developer.gnome.org/glib/unstable/glib-Testing.html

Libqtest

- API to control QEMU
- Expands GLib test framework:
  - Wraps QEMU init
  - Enables debugging functions
  - Performs a clean exit
- Adds basic operations:
  - Clock
  - Memory and I/O
  - IRQ
  - QMP (QEMU machine protocol)

LibQOS

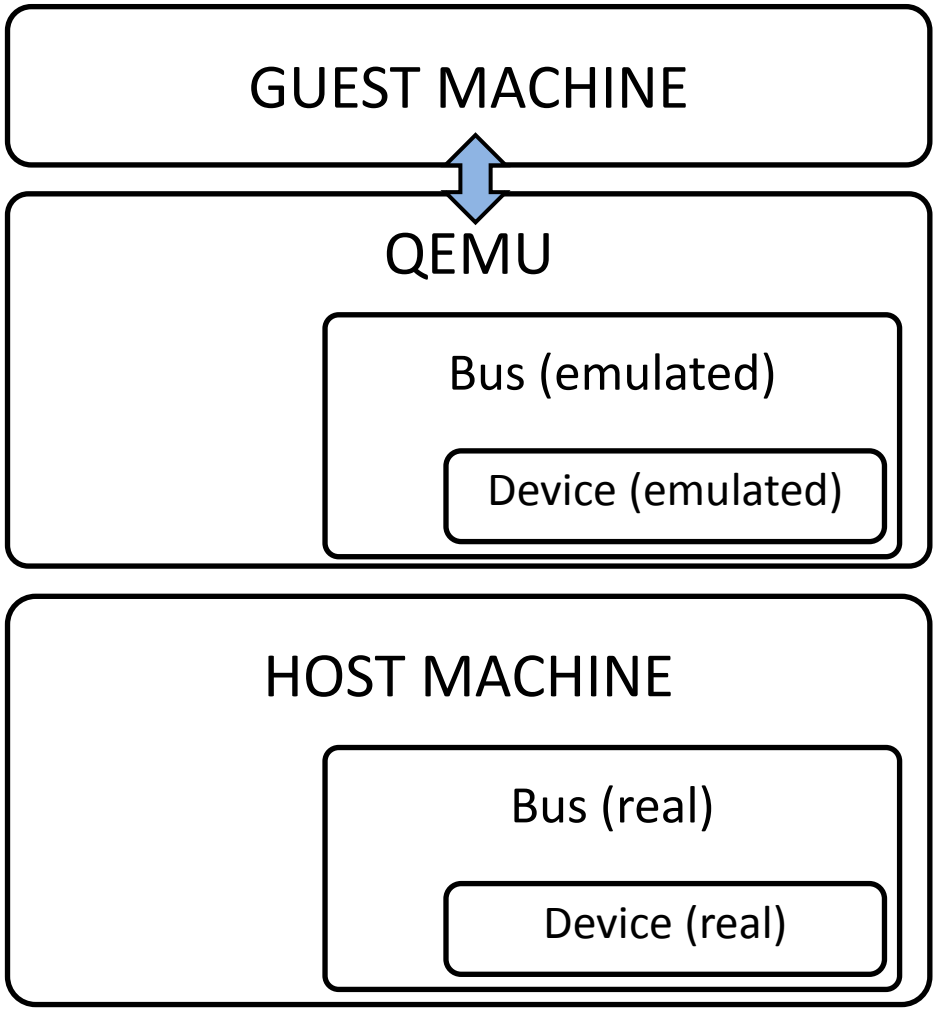
- Device driver framework for writing qtest cases
- Bus wrappers
- Contains functions specific to each bus
- Simplifies the device developer work
- Standarizes access to devices

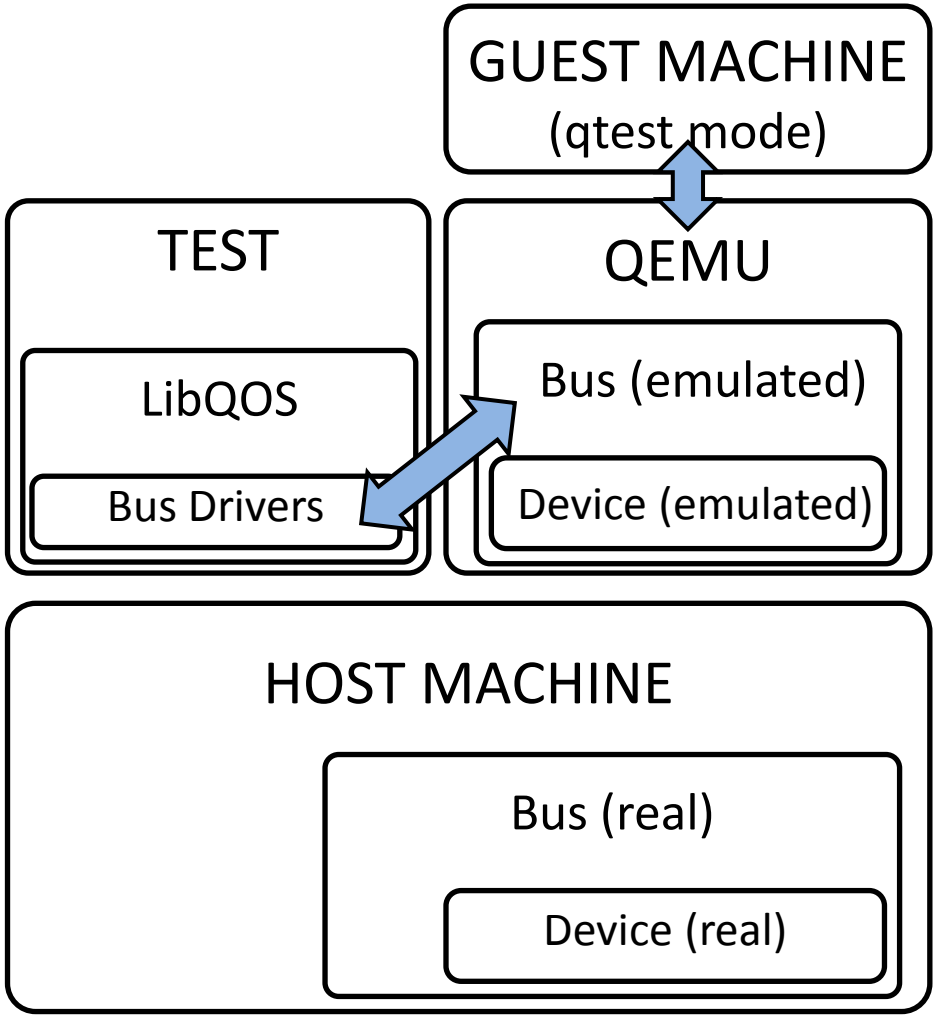
Objective

- Have a complete test suite
- Each device implemented has one test suite
- LibQOS has a implementation for each bus
- Create a full testing enviroment:
  - Can detect loaded devices
  - Can check automatically and autonomously

Source: http://www.linux-kvm.org/wiki/images/8/89/2012-forum-Liguori-qtest.pdf

Index


GUEST MACHINE
QEMU
Bus (emulated)
Device (emulated)
HOST MACHINE
Bus (real)
Device (real)


GUEST MACHINE
(qtest mode)
TEST
LibQOS
Bus Drivers
QEMU
Bus (emulated)
Device (emulated)
HOST MACHINE
Bus (real)
Device (real)

Index

Simple test case

```
/* AC97 test case */
static void nop(void) { }
int main(int argc, char **argv)
{
    int ret;
    g_test_init(&argc, &argv, NULL);
    qtest_add_func("/ac97/nop", nop);
    qtest_start("-device AC97");
    ret = g_test_run();
    qtest_end();
    return ret;
}
```

Simple test case

```
/* AC97 test case */
static void nop(void) { }
int main(int argc, char **argv)
{
    int ret;
    g_test_init(&argc, &argv, NULL);
    qtest_add_func("/ac97/nop", nop);
    qtest_start("-device AC97");
    ret = g_test_run();
    qtest_end();
    return ret;
}
```

Initialize the GLib testing framework

Simple test case

```
/* AC97 test case */
static void nop(void) { }
int main(int argc, char **argv)
{
    int ret;
    g_test_init(&argc, &argv, NULL);
    qtest_add_func("/ac97/nop", nop);
    qtest_start("-device AC97");
    ret = g_test_run();
    qtest_end();
    return ret;
}
```

Add the test case with path `/ac97/nop` and function `nop`

g_test_add_func() or qtest_add_func()

qtest_add_func() adds the architecture in front of the path:

```
qtest_add_func("/ac97/nop", nop);
```

Is equivalent to (running a i386 guest):

```
g_test_add_func("/i386/ac97/nop", nop);
```

Simple test case

```
/* AC97 test case */
static void nop(void) { }
int main(int argc, char **argv)
{
    int ret;
    g_test_init(&ar
    qtest_add_func("/ac97/nop", nop);
    qtest_start("-device AC97");
    ret = g_test_run();
    qtest_end();
    return ret;
}
```

Setup and start the guest machine with the extra QEMU parameters –device AC97

Simple test case

```
/* AC97 test case */
static void nop(void) { }
int main(int argc, char **argv)
{
    int ret;
    g_test_init(&argc, &argv, NULL);
    qtest_add_func("/ac97/nop", nop);
    qtest_start("-device AC97");
    ret = g_test_run();
    qtest_end();
    return ret;
}
```

Run the test and perform a clean exit

Libqtest API – IRQ

```
/* ide-test.c extract */
irq_intercept_in("ioapic");
/* More test here */
g_assert(!get_irq(14));
```

Also `void irq_intercept_out(const char *string)`

Libqtest API – QMP

```
/* qdev-monitor-test.c extract */
response = qmp("{\"execute\": \"device_add\","
               " \"arguments\": {"
               "   \"driver\": \"virtio-blk-pci\","
               "   \"drive\": \"drive0\""
               "}}");
g_assert(response);
error = qdict_get_qdict(response, "error");
g_assert_cmpstr(
     qdict_get_try_str(error, "class"), ==,
     "GenericError");
QDECREF(response);
```

Libqtest API – QMP

```
/* virtio-blk-test.c extract */
qmp_discard_response("{"
    "'execute': 'block_resize', "
    "'arguments': { 'device': 'drive0', "
                  "'size': %d "
    "}}", n_size);
```

Libqtest API – Clock

```
/* rtc-test.c extract */
for (i = 0; i < 4; i++) {
  if (get_irq(RTC_ISA_IRQ)) {
     break;
  }
  clock_step(1000000000);
}
```

Also `int64_t clock_step_next(void)`

`int64_t clock_set(int64_t val)`

Libqtest API – Memory

To read and write from the guest memory:

```
uint8_t readb(uint64_t addr)
uint16_t readw(uint64_t addr)
uint32_t readl(uint64_t addr)
uint64_t readq(uint64_t addr)
void memread(uint64_t addr, void *data, size_t size)

void writeb(uint64_t addr, uint8_t value)
void writew(uint64_t addr, uint16_t value)
void writel(uint64_t addr, uint32_t value)
void writeq(uint64_t addr, uint64_t value)
void memwrite(uint64_t addr, const void *data, size_t size)

void qmemset(uint64_t addr, uint8_t patt, size_t size)
```

Libqtest API – I/O

To read and write from I/O space:

```
uint8_t inb(uint64_t addr)
uint16_t inw(uint64_t addr)
uint32_t inl(uint64_t addr)

void outb(uint64_t addr, uint8_t value)
void outw(uint64_t addr, uint16_t value)
void outl(uint64_t addr, uint32_t value)
```

Libqtest API – Misc

```
/* virtio-blk-test.c extract */
const char *arch = qtest_get_arch();

if (strcmp(arch, "i386") == 0 ||
                strcmp(arch, "x86_64") == 0) {
    qtest_add_func("/virtio/blk/pci/basic",
                                    pci_basic);
} else if (strcmp(arch, "arm") == 0) {
    qtest_add_func("/virtio/blk/mmio/basic",
                                  mmio_basic);
}
```

Libqtest API – Misc

```
/* libqos/virtio-pci.c extract */
if (qtest_big_endian()) {
        for (i = 0; i < 8; ++i) {
            u64 |= (uint64_t)qpci_io_readb(
                        dev->pdev, addr + i)
                        << (7 - i) * 8;
        }
    } else {
        for (i = 0; i < 8; ++i) {
            u64 |= (uint64_t)qpci_io_readb(
                        dev->pdev, addr + i)
                        << i * 8;
        }
    }
```

Index

Guest memory functionalities

- Allocate memory: `qguest_alloc`
- Free memory: `qguest_free`

PCI functionalities

- Device operations (`qpci_device_find`, `qpci_device_enable`…)
- Config operations (`qpci_config_readb`, `qpci_config_writel`…)
- I/O operations (`qpci_iomap`, `qpci_io_readw`, `qpci_io_writeb`…)
- MSIX functionalities (`qpci_msix_enable`, `qpci_msix_pending`…)

VirtIO functionalities

- Device operations (`qvirtio_pci_device_enable`, `qvirtio_set_features`…)
- Config operations (`qvirtio_config_readb`, `qvirtio_config_writel`…)
- Virtqueues (`qvirtqueue_setup`, `qvirtqueue_add`…)
- Interruptions (`qvirtio_wait_queue_isr`…)

Index

Where's the code?

- Tests: qemu/tests/
- Libqos drivers: qemu/tests/libqos/
- Makefile: qemu/tests/Makefile

How to add a test

```
/* Makefile extract */
tests/usb-hcd-ehci-test$(EXESUF): \
      tests/usb-hcd-ehci-test.o $(libqos-pc-obj-y)
tests/vhost-user-test$(EXESUF): \
      tests/vhost-user-test.o qemu-char.o \
      qemu-timer.o $(qtest-obj-y)
tests/qemu-iotests/socket_scm_helper$(EXESUF): \
      tests/qemu-iotests/socket_scm_helper.o
tests/test-qemu-opts$(EXESUF): \
      tests/test-qemu-opts.o libqemuutil.a \
      libqemustub.a
tests/new-test$(EXESUF): tests/new-test.o \
      {dependencies}
```

Compiling and running tests

- Compile and run all the test suite:

```
make check
```

- Compile just your test:

```
make tests/new-test
```

- Run your test:

```
QTEST_QEMU_BINARY=\
    i386-softmmu/qemu-system-i386 \
    tests/new-test
```

Debugging

```
QTEST_LOG=1 QTEST_STOP=1 \
    QTEST_QEMU_BINARY=\
    i386-softmmu/qemu-system-i386 \
    tests/new-test
```

Debugging

- QTEST_LOG=1: write to stderr all operations

```
[R +0.025815] outl 0xcf8 0x80000000
[S +0.025852] OK
[R +0.025881] inw 0xcfc
[S +0.025900] OK 0x8086
[R +0.025927] outl 0xcf8 0x80000000
[S +0.025940] OK
[R +0.025963] inw 0xcfc
[S +0.025974] OK 0x8086
```

Debugging

- QTEST_STOP=1: stop to connect the debugger
  - Attach GDB:

```
gdb --pid=$(pidof new-test)
```

  - Continue executing:

```
kill -SIGCONT \
        $(pidof qemu-system-i386)
```

Index

Conclusion

- Testing in QEMU is essential to maintain integrity
- Libqtest and libqos make developing device tests in QEMU easier.
- There is a lack of tests for devices

Have fun coding them!

Thanks to

- Stefan Hajnoczi
- Paolo Bonzini
- All the QEMU people that is open to questions every day at any hour

Questions?