KVM without QEMU

Gabriel Laskar <gabriel@lse.epita.fr>

Agenda

- What is kvm ?
- What we want to achieve ?
- kvm api overview
- how to start in 32-bit
- virtio device
- machine description
- state of the tool

What is KVM ?

- Full virtualization solution for Linux
- splitted in 2 parts :
  - Kernel module
  - Userland application
    - QEMU
    - lkvm

What are we trying to achieve ?

Create a virtual machine :

- without any backward compatibility stuff
- virtio drivers
- simple access to virtualized hardware
- ...

Goals

- Simple VM for experimentation
- Thin specialized VMs
- Because we can !

Documentation

It exists !

- <linux>/Documentation/virtual/kvm/api.txt
- kvm code
- qemu code
- lkvm code

KVM API Overview

- /dev/kvm char device
- ioctls for requests
- 1 fd per resource :
    - system : vm creation, capabilities
    - vm : cpu creation, memory, irq
    - vcpu : access to state

VM Creation

```c
int fd_kvm = open("/dev/kvm", O_RDWR);

int fd_vm = ioctl(fd_kvm, KVM_CREATE_VM, 0);

// add space for some strange reason on intel (3 pages)
ioctl(fd_vm, KVM_SET_TSS_ADDR, 0xffffffffffffd000);

ioctl(fd_vm, KVM_CREATE_IRQCHIP, 0);
```

Add Physical Memory

```
// set memory region
void *addr = mmap(NULL, 10 * MB, PROT_READ | PROT_WRITE,
    MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);

struct kvm_userspace_memory_region region = {
    .slot = 0,
    .flags = 0, // Can be Read Only
    .guest_phys_addr = 0x100000,
    .memory_size = 10 * MB,
    .userspace_addr = (__u64)addr
};

ioctl(fd_vm, KVM_SET_MEMORY_REGION, &region);
```

Create a VCPU and Initialisation

```
int fd_vcpu = ioctl(fd_vm, KVM_CREATE_VCPU, 0);

struct kvm_regs regs;
ioctl(fd_vcpu, KVM_GET_REGS, &regs);
regs.rflags = 0x02;
regs.rip = 0x0100f000;
ioctl(fd_vcpu, KVM_SET_REGS, &regs);
```

Running a VCPU

```c
int kvm_run_size = ioctl(fd_kvm, KVM_GET_VCPU_MMAP_SIZE, 0);

// access to the arguments of ioctl(KVM_RUN)
struct kvm_run *run_state =
    mmap(NULL, kvm_run_size, PROT_READ | PROT_WRITE,
       MAP_PRIVATE, fd_vcpu, 0);


for (;;) {
    int res = ioctl(fd_vcpu, KVM_RUN, 0);

    switch (run_state->exit_reason) {
        // use run_state to gather informations about the exit
    }
}
```

Devices : PIO

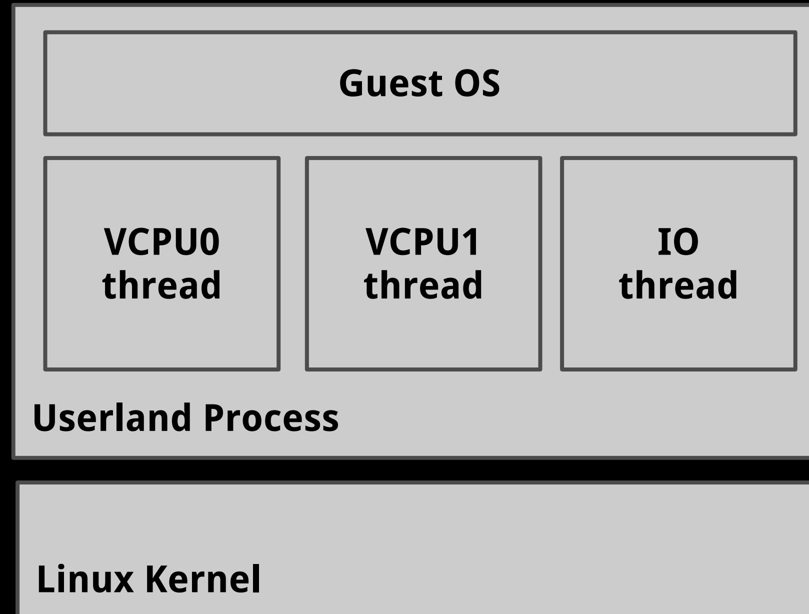
```
switch (run_state->exit_reason) {
    case KVM_EXIT_IO:
        if (run_state->io.port == CONSOLE_PORT
         && run_state->io.direction == KVM_EXIT_IO_OUT) {

            __u64 offset = run_state->io.data_offset;
            __u32 size = run_state->io.size;

            write(STDOUT_FILENO, (char*)run_state + offset, size);
        }
    break;
    // ...
}
```

Devices : MMIO

Exit reason : KVM_EXIT_MMIO

```
struct {
  __u64 phys_addr;
  __u8  data[8];
  __u32 len;
  __u8  is_write;
} mmio;
```

Organisation of the VMM



How to start in Protected Mode

```c
void vcpu_pm_mode(struct vcpu *vcpu) {
    struct kvm_sregs sregs;
    vcpu_get_sregs(vcpu, sregs);

    // setup basic flat model
    sregs.cs.base = 0x0; sregs.cs.limit = 0xffffffff; sregs.cs.g = 1;
    sregs.ds.base = 0x0; sregs.ds.limit = 0xffffffff; sregs.ds.g = 1;
    sregs.ss.base = 0x0; sregs.ss.limit = 0xffffffff; sregs.ss.g = 1;

    // set default operation size and stack pointer size to 32-bit
    sregs.cs.db = 1;
    sregs.ss.db = 1;

    // activate PM bit in cr0
    sregs.cr0 |= 0x01;

    vcpu_set_sregs(vcpu, sregs);
}
```

Devices

- Configuration via MMIO/PIO
- eventfd for events between host/guest
  - irqfd : host → guest
  - ioeventfd : guest → host

What is virtio

- Abstraction for virtualized devices
- spec available
- standardisation in progress
- 2 types of devices : pci or mmio
- configuration
- queues

How to advertise device configuration ?

Existing standards :

- ACPI
- MP tables
- PCI
- SFI

All these choices are complex, or old.

Solution : create our own structure and give it to the kernel

Machine informations

```
struct start_info {
    uint ioapic_base;
    uint mem_size;
    uint mem_entries;
    uint dev_size;
    uint dev_entries;
};
```

```
struct memory_map_entry {
    uint  base;
    uint  size;
#define MEMORY_FLAG_READ_ONLY  1
    uint  flags;
#define MEMORY_USE_FREE   0
#define MEMORY_USE_KERNEL 1
#define MEMORY_USE_INIT   2
    uint  use;
};

struct device_map_entry {
    uint base_addr;
    uint interrupt_num;
};
```

State of the Proof of Concept

- start in PM
- load an ELF binary
- simple struct passed to describe devices
- Started Virtio mmio configuration
- Virtio queues
- virtio-rng device

Thank you

- gabriel@lse.epita.fr
- http://www.linux-kvm.org
- http://qemu.org
- https://github.com/penberg/linux-kvm
- https://github.com/rustyrussell/virtio-spec