使用KVM API实现Emulator Demo

这边文章来描述如何用KVM API来写一个Virtualizer的demo code, 也就是相当与Qemu，用来做设备模拟。 此文是帮助想了解KVM原理已经Qemu原理的人 or Just for fun.

完整的Code在这里： https://github.com/soulxu/kvmsample

这个code其实是很久以前写的，以前在team内部分享过，用来帮助大家理解kvm工作原理。现在既然要开始写code了，就用这个先来个开端。

当然我不可能写一个完整的Qemu，只是写出Qemu中最基本的那些code。这个虚拟机只有一个VCPU和512000000字节内存(其实富裕了) 可以进行一些I/O，当然这些I/O的结果只能导致一些print，没有实际模拟任何设备。所以所能执行的Guest也很简单。

首先来看看Guest有多简单。

```
1 .globl _start
2     .code16
3 _start:
4     xorw %ax, %ax
5
6 loop1:
7     out %ax, $0x10
8     inc %ax
9     jmp loop1
```

不熟悉汇编也没关系，这code很简单，基本也能猜到干啥了。对，Guest只是基于at&t汇编写的一个在8086模式下的死循环，不停的向端口0x10写东西。目标就是让这个Guest跑起来了。

我们的目标就是让这个Guest能执行起来。下面开始看我们虚拟机的code了。

我们先来看看main函数：

```
1  int main(int argc, char **argv) {
2      int ret = 0;
3      struct kvm *kvm = kvm_init();
4
5      if (kvm == NULL) {
6          fprintf(stderr, "kvm init faulit\n");
7          return -1;
8      }
9
10     if (kvm_create_vm(kvm, RAM_SIZE) < 0) {
11         fprintf(stderr, "create vm fault\n");
12         return -1;
13     }
14
15     load_binary(kvm);
16
17     // only support one vcpu now
18     kvm->vcpu_number = 1;
19     kvm->vcpus = kvm_init_vcpu(kvm, 0, kvm_cpu_thread);
20
21     kvm_run_vm(kvm);
22
23     kvm_clean_vm(kvm);
24     kvm_clean_vcpu(kvm->vcpus);
25     kvm_clean(kvm);
26 }
```

这里正是第一个kvm基本原理： 一个虚拟机就是一个进程，我们的虚拟机从这个main函数开始

让我先来看看kvm_init。这里很简单，就是打开了/dev/kvm设备，这是kvm的入口，对kvm的所有操作都是通过对文件描述符上执行ioctl来完成。 这里很简单，就是打开kvm设备，然后将文件描述符返回到我自己创建的一个结构体当中。

然后我们就开始创建一个vm，然后为其分配内存。

```
1 kvm->vm_fd = ioctl(kvm->dev_fd, KVM_CREATE_VM, 0);
```

创建一个虚拟机很简单，在kvm设备上执行这么一个ioctl即可，然后会得到新建的vm的文件描述，用来操作这个vm。

然后我们来分配内存，这里最重要的是struct kvm_userspace_memory_region这个数据结构。

```
1 /* for KVM_SET_USER_MEMORY_REGION */
2 struct kvm_userspace_memory_region {
3         __u32 slot;
4         __u32 flags;
5         __u64 guest_phys_addr;
6         __u64 memory_size; /* bytes */
7         __u64 userspace_addr; /* start of the userspace allocated memory */
8 };
```

memory_size是guest的内存的大小。userspace_addr是你为其份分配的内存的起始地址，而guest_phys_addr则是这段内存映射到guest的什么物理内存地址。

这里用mmap创建了一段匿名映射，并将地址置入userspace_addr。随后来告诉我们的vm这些信息：

```
1 ioctl(kvm->vm_fd, KVM_SET_USER_MEMORY_REGION, &(kvm->mem));
```

这里是来操作我们的vm了，不是kvm设备文件了。

我们有了内存了，现在可以把我们的guest code加载的进来了，这个实现很简单就是打开编译后的二进制文件将其写入我们分配的内存空间当中。 这里所要注意的就是如何编译guest code，这里我们编译出来的是flat binary，不需要什么elf的封装。

有了内存，下一步就是vcpu了，创建vcpu是在kvm_init_vcpu函数里。 这里最重要的操作只有这个：

```
1 vcpu->kvm_run_mmap_size = ioctl(kvm->dev_fd, KVM_GET_VCPU_MMAP_SIZE, 0);
2 ...
3 vcpu->kvm_run = mmap(NULL, vcpu->kvm_run_mmap_size, PROT_READ | PROT_WRITE, MAP_SHARED, vcpu->vcpu_fd, 0);
```

struct kvm_run是保存vcpu状态的一个数据结构，稍后我们可以看到我们可以从这里得到当陷入后具体陷入原因。

有了内存和vcpu就可以运行了：

```
1 pthread_create(&(kvm->vcpus->vcpu_thread), (const pthread_attr_t *)NULL, kvm->vcpus[i].vcpu_thread_func, kvm)
```

这里是另一个kvm基本概念了，一个vcpu就是一个线程。这里让我们为vcpu创建一个线程。

最终我们到了最关键的部分了，就是这个vcpu线程。其实他就是一个循环。 当循环开始的时候，我们让他执行guest code:

```
1 ret = ioctl(kvm->vcpus->vcpu_fd, KVM_RUN, 0)
```

当执行这条语句后，guest code就开始执行了，这个函数就阻塞在这里了。直到something happened而且需要由hypervisor进行处理的时候这个函数才会返回。 比如说I/O发生了，这个函数就会返回了，这里我们就需要通过struct kvm_run中得到具体的陷入原因。我们的guest只是做一些I/O port的操作，所以可以看到 当退出原因是KVM_EXIT_IO时，我将guest的所写入的数据print出来。

到这里这就是这个virtualizer的全部了. 如果你想体验一下，只需要执行make。

```
1 :~/code/kvmsample$ make
2 cc    -c -o main.o main.c
3 gcc main.c -o kvmsample -lpthread
4 as -32 test.S -o test.o
5 ld -m elf_i386 --oformat binary -N -e _start -Ttext 0x10000 -o test.bin test.o
```

然后执行kvmsample

```
1  $ ./kvmsample
2  read size: 712288
3  KVM start run
4  KVM_EXIT_IO
5  out port: 16, data: 0
6  KVM start run
7  KVM_EXIT_IO
8  out port: 16, data: 1
9  KVM start run
10 KVM_EXIT_IO
11 out port: 16, data: 2
12 KVM start run
13 KVM_EXIT_IO
14 out port: 16, data: 3
15 ....
```

其实qemu里面的code也就是这样，你也可以在其中找到这个loop，只不过它被qemu内部的各种设备框架所隐藏起来了。

Posted by Alex Xu (soulxu@gmail.com) virtualization

Some ideas for Nova API in Kilo »

About Me

- Alex Xu
- Email: soulxu@gmail.com
- Weibo: @徐小杰子

Recent Posts