实验一: CPU 虚拟化

[522031910557] [冯海桐]

1 调研部分

1. 通常将所有敏感指令都是特权指令的架构称为可虚拟化架构，反之存在敏感非特权指令的架构称为不可虚拟化架构。敏感指令是指操作敏感物理资源的指令，特权指令是指必须运行在最高特权级的指令。在虚拟化环境中，Hypervisor处于最高特权级，监控所有特权指令，如果存在敏感非特权指令，用户就有可能绕过Hypervisor直接访问敏感物理资源，从而出现严重的问题。所以，存在敏感非特权指令的架构不能直接虚拟化，需要借助其他软硬件解决方案。
2. Hypervisor处于最高特权限，虚拟机对物理资源的访问触发异常，陷入Hypervisor之中，受到Hypervisor的监控和模拟。“陷入再模拟” 这一过程保证了用户无需知晓运行的具体物理环境，一切的特权指令都将交由Hypervisor执行；同时，也方便Hypervisor统一调度多个虚拟机的特权指令，有效防止了物理资源的使用出现问题。
3. Intel VT-x引入了VMX操作模式，其包含根模式和非根模式，两个模式都有各自的四个特权级（Ring0~Ring3）。虚拟机操作系统和应用程序分别运行再非根模式的Ring0和Ring3特权级中，而Hypervisor通常运行再根模式的Ring0特权级，三者的特权级划分问题就此解决。在非根模式下，虚拟机执行特权指令时会触发VM Exit，将控制权交给Hypervisor。这种机制确保了虚拟机无法直接执行特权指令，必须通过Hypervisor来处理。

2 实验目的

1. 理解虚拟化的基本概念；
2. 理解CPU及中断虚拟化的基本原理；
3. 编写 “陷入再模拟” 过程，完成相关分支。

3 实验步骤

1. 启动并登陆到 L1 虚拟机中。
2. 进入 L1 虚拟机的 cpu_lab 目录下：`$ cd ~/labs/cpu_lab` 。
3. 根据源代码用 `vim` 对 `sample-qemu.c` 文件进行补全。完成`case KVM_EXIT_IO` 分支，使得 VM 往端口写入的 “Hello World!” 字符串变为小写后打印到标准输出中。
4. 通过 `$ make` 编译目标程序，运行后可以在终端中看到输出结果。

4 实验分析

4.1 代码理解

4.1.1 初始化和设置

`open("/dev/kvm", O_RDWR | O_CLOEXEC)`： 以读写模式 `O_RDWR` 和执行时关闭文件描述符 `O_CLOEXEC` 模式打开 `/dev/kvm` 设备文件。

`ioctl(kvm, KVM_GET_API_VERSION, NULL)`：通过 `ioctl` 系统调用获取 KVM 的 API 版本。如果调用失败或者 API 版本不是 12，输出错误信息并退出程序。

`ioctl(kvm, KVM_CREATE_VM, (unsigned long)0)`：通过 `ioctl` 系统调用创建一个新的虚拟机实例。

`mmap(NULL, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0)`：分配一页对齐具有读写权限的内存，用于存放虚拟机的代码。

`ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &region)`：通过 `ioctl` 系统调用设置虚拟机的用户内存区域。

`ioctl(vmfd, KVM_CREATE_VCPU, (unsigned long)0)`：通过 `ioctl` 系统调用创建一个虚拟 CPU.

`ioctl(kvm, KVM_GET_VCPU_MMAP_SIZE, NULL)`：通过 `ioctl` 系统调用获取 VCPU 的 mmap 大小。

`run = mmap(NULL, mmap_size, PROT_READ | PROT_WRITE, MAP_SHARED, vcpufd, 0)`：映射共享的 `kvm_run` 结构和后续数据。

`ioctl(vcpufd, KVM_GET_SREGS, &sregs)`：通过 `ioctl` 系统调用获取 VCPU 的段寄存器。

`ioctl(vcpufd, KVM_SET_SREGS, &sregs)`：通过 `ioctl` 系统调用设置 VCPU 的段寄存器。

`ioctl(vcpufd, KVM_SET_REGS, &regs)`：通过 `ioctl` 系统调用将 `regs` 结构体中的值设置到 VCPU 的寄存器中。

4.1.2 运行虚拟机

`ioctl(vcpufd, KVM_RUN, NULL)`：通过 ioctl 系统调用运行 VCPU。

`switch (run->exit_reason)`：根据虚拟机退出的原因进行分支处理。

`KVM_EXIT_HLT`：虚拟机执行了 `HLT` （休眠）指令，输出 "KVM_EXIT_HLT" 并返回 0，结束程序。

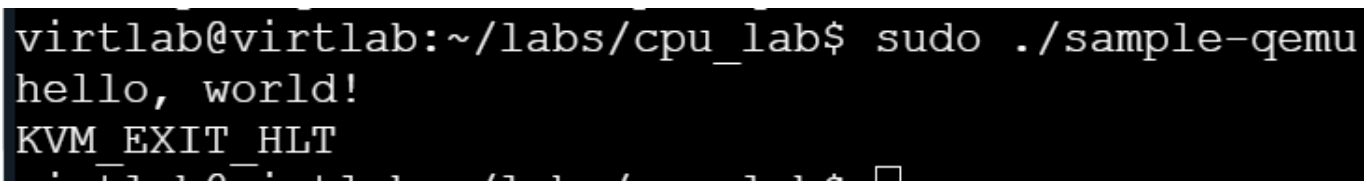
`KVM_EXIT_IO`：虚拟机执行了 I/O 操作。检查 I/O 操作是否是向端口 0x3f8 输出一个字节的数据，且操作数是否为一次。如果是，获取要输出的字符并将大写字母转为小写并输出；如果否，输出错误信息并退出。

`KVM_EXIT_FAIL_ENTRY`：出现硬件进入失败，输出错误信息并退出程序。

`KVM_EXIT_INTERNAL_ERROR`：出现内部错误，输出错误信息并退出程序。

4.2 实验结果

补全代码后的运行输出如下图所示：



正确输出了结果。

5 遇到的问题及解决方案

第一次执行的时候，使用的是 `$ ./sample-qemu` ，结果报错 `sample-qemu: /dev/kvm: Permission denied` ，经过查询后发现这个错误信息表明当前用户没有权限访问 `/dev/kvm` 设备文件。使用 `sudo` 后成功解决。

附录

```c
#include <err.h>
#include <fcntl.h>
#include <linux/kvm.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/types.h>

int main(void)
{
    int kvm, vmfd, vcpufd, ret;
    const uint8_t code[] = {
        0xba, 0xf8, 0x03, /* mov $0x3f8, %dx */
        0xb0, 'H',        /* mov $'H', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'e',        /* mov $'e', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'l',        /* mov $'l', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'l',        /* mov $'l', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'o',        /* mov $'o', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, ',',        /* mov $',', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, ' ',        /* mov $' ', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'w',        /* mov $'w', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'o',        /* mov $'o', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'r',        /* mov $'r', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'l',        /* mov $'l', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, 'd',        /* mov $'d', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, '!',        /* mov $'!', %al */
        0xee,              /* out %al, (%dx) */
        0xb0, '\n',        /* mov $'\n', %al */
        0xee,              /* out %al, (%dx) */
        0xf4,              /* hlt */
    };
    uint8_t *mem;
    struct kvm_sregs sregs;
    size_t mmap_size;
```

```
    struct kvm_run *run;

    kvm = open("/dev/kvm", O_RDWR | O_CLOEXEC);
    if (kvm == -1)
        err(1, "/dev/kvm");

    /* Make sure we have the stable version of the API */
    ret = ioctl(kvm, KVM_GET_API_VERSION, NULL);
    if (ret == -1)
        err(1, "KVM_GET_API_VERSION");
    if (ret != 12)
        errx(1, "KVM_GET_API_VERSION %d, expected 12", ret);

    vmfd = ioctl(kvm, KVM_CREATE_VM, (unsigned long)0);
    if (vmfd == -1)
        err(1, "KVM_CREATE_VM");

    /* Allocate one aligned page of guest memory to hold the code. */
    mem = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS,
-1, 0);
    if (!mem)
        err(1, "allocating guest memory");
    memcpy(mem, code, sizeof(code));

    /* Map it to the second page frame (to avoid the real-mode IDT at 0). */
    struct kvm_userspace_memory_region region = {
        .slot = 0,
        .guest_phys_addr = 0x1000,
        .memory_size = 0x1000,
        .userspace_addr = (uint64_t)mem,
    };
    ret = ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &region);
    if (ret == -1)
        err(1, "KVM_SET_USER_MEMORY_REGION");

    vcpufd = ioctl(vmfd, KVM_CREATE_VCPU, (unsigned long)0);
    if (vcpufd == -1)
        err(1, "KVM_CREATE_VCPU");

    /* Map the shared kvm_run structure and following data. */
    ret = ioctl(kvm, KVM_GET_VCPU_MMAP_SIZE, NULL);
    if (ret == -1)
        err(1, "KVM_GET_VCPU_MMAP_SIZE");
    mmap_size = ret;
    if (mmap_size < sizeof(*run))
        errx(1, "KVM_GET_VCPU_MMAP_SIZE unexpectedly small");
    run = mmap(NULL, mmap_size, PROT_READ | PROT_WRITE, MAP_SHARED, vcpufd, 0);
    if (!run)
        err(1, "mmap vcpu");

    /* Initialize CS to point at 0, via a read-modify-write of sregs. */
    ret = ioctl(vcpufd, KVM_GET_SREGS, &sregs);
    if (ret == -1)
        err(1, "KVM_GET_SREGS");
```

```
    sregs.cs.base = 0;
    sregs.cs.selector = 0;
    ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
    if (ret == -1)
        err(1, "KVM_SET_SREGS");

    /* Initialize registers: instruction pointer for our code, addends, and
     * initial flags required by x86 architecture. */
    struct kvm_regs regs = {
        .rip = 0x1000,
        .rax = 2,
        .rbx = 2,
        .rflags = 0x2,
    };
    ret = ioctl(vcpufd, KVM_SET_REGS, &regs);
    if (ret == -1)
        err(1, "KVM_SET_REGS");

    /* Repeatedly run code and handle VM exits. */
    while (1) {
        ret = ioctl(vcpufd, KVM_RUN, NULL);
        if (ret == -1)
            err(1, "KVM_RUN");
        switch (run->exit_reason) {
        case KVM_EXIT_HLT:
            puts("KVM_EXIT_HLT");
            return 0;
        case KVM_EXIT_IO:
        // ---------------------------------------
        //---------- START YOUR CODE -------------
            if (run->io.direction == KVM_EXIT_IO_OUT && run->io.size == 1 && run-
>io.count == 1 && run->io.port == 0x3f8) {
                char *p = (char *)run;
                p += run->io.data_offset;
                char ch = *p;
                if (ch >= 'A' && ch <= 'Z') {
                    ch += 'a' - 'A';
                }
                putchar(ch);
                fflush(stdout);
            } else {
                errx(1, "IO operation did not meet the required conditions for
printing data.");
            }
            break;
        // --------- END OF YOUR CODE ------------
        //  ---------------------------------------
        case KVM_EXIT_FAIL_ENTRY:
            errx(1, "KVM_EXIT_FAIL_ENTRY: hardware_entry_failure_reason = 0x%llx",
                 (unsigned long long)run-
>fail_entry.hardware_entry_failure_reason);
        case KVM_EXIT_INTERNAL_ERROR:
            errx(1, "KVM_EXIT_INTERNAL_ERROR: suberror = 0x%x", run-
>internal.suberror);
```

```
            default:
                errx(1, "exit_reason = 0x%x", run->exit_reason);
            }
        }
    }
```