HOMEWORK 3

Welcome to your homework, here are the following are the **rules** of submitting homework:

- You have to submit the code/report in **PDF**, compiled from **Markdown**. Any report written in word document or so will be discarded and score 0.
- You have to submit your homework via Email to ncuhomework@outlook.com.
- Your Email subject shall be ***Course Z6110X0035 CC+Homework #+{Your Name}+{Your ID}***. For example, if I submit a homework, my Email subject is Course Z6110X0035 CC Homework 3 Zichen Xu 1234556678.

Assignment

Q1: Write a hypercall in KVM

A hypercall is a way for a guest OS to make a call to the hypervisor, in some ways similar to how a system call allows an application to make a call to the OS. We are asking you to write a hypercall to become familiar with how they work and the codebase for KVM. For this part of the assignment, you will also set up your VM so you can use it for your hypercall development.

The hypercall you write should take one argument and output the information about virtual CPU in KVM. Modern architectures provide special support for virtualization and add a privileged instruction for hypercalls, which you will use in this part of the assignment. Rather than using the default kernel that comes with Ubuntu 14.04.5, you should download and build a more recent version of the Linux kernel, kernel version 4.10, and install that in your VM. It will be this updated version of the kernel/KVM source that you will modify to implement your hypercall.

The prototype for your hypercall will be the following. The argument vcpu_id contains the CPU id in your VM.

```
int vcpu_info(int vcpu_id);
```

Your hypervisor should output the following information to ftrace in your KVM host based on the input VCPU ID provided by the VM.

```
pid: the corresponding PID of the VCPU thread in KVM host
gp_regs: the values of the general purpose registers for the virtual CPU
num_exits: number of vm exits from the VCPU
```

The followings are other features you need to implement in your hypercall. Your code should handle errors properly. At a minimum, your hypercall should detect and report the condition which the virtual CPU id passed by the VM is not valid and return error code to the VM. You should assign an unique number for your hypercall, so KVM can identify the call and handle it properly.

HINT: Intel's (most likely CPU you have in your personal computer) virtualization extension (VMX) provides a "privileged" instruction called vmcall for hypercall. When the VM executes the instruction, it traps from non-root operation to root operation so the hypervisor can then handle the hypercall. You should search the Linux source and look at how vmcall is handled by KVM.

HINT: You can use the Linux Cross-Reference (LXR) to investigate different hypercalls already defined. You should use the same calling convention as the other hypercalls.

HINT: You should look at how the structure kvm_vcpu is defined and used in the source code.

Compiling / Updating Linux/KVM host:

You can download the Linux v4.10 mainline kernel source via:

```
git clone https://github.com/torvalds/linux.git -b v4.10
```

In your Linux source directory, first copy the ubuntu config to your kernel source. Note that this only has to be done once.

```
cp /boot/config-#YOUR_UBUNTU_VERSION .config
yes "" | make oldconfig
```

Install proper packages for the kernel compile.

```
sudo apt-get install libssl-dev bc libncurses-dev
```

Run the following command to configure your kernel.

```
make menuconfig
```

To enable ftrace in your KVM host, go to **Kernel Hacking** and select **Tracers**. Then enter **Tracers**, select both **Kernel Function Tracer** and **Kernel Function Graph Tracer**. Finally, save the config and exit.

After you add your hypercall to your Linux/KVM source, use the following command to compile the kernel.

```
make -j8
```

Finally, use the following command to install the new kernel to your system. The new kernel will be loaded in the next boot.

```
make modules_install
make install
```

Q2: Test your new hypercall

To test your new hypercall, you will install your modifications in the guest kernel running in your nested VM for testing. As mentioned earlier, the instruction that initiates a hypercall is a privileged instruction and cannot be executed in user mode. To test the hypercall from user space, you need to add a new system call in your guest kernel which in turn calls to the vcpu_info. The prototype of the system call can be defined as the following.

```
int sys_vcpu_info(void);
```

Your sys*vcpu*info system call should enumerate each of the online CPU, and pass the CPU id to KVM via the vcpu_info hypercall.

You should write a simple C program which calls to sys*vcpu*info from user space. You can get the see the output from the hypercall in your KVM host using the following command:

```
cat /sys/kernel/debug/tracing/trace
```

NOTE: Although system calls are generally accessed through a library (libc), your test program should access your system call directly. This is accomplished by utilizing the general purpose syscall(2) system call (consult its man page for more details).

IMPORTANT NOTE:

- Submit a report of your procedure running your hypercall.
- Submit your code file of hypercall
- Submit a test report of your hypercall while running different test cases (correct/incorrect inputs).