CSIE 5310 Assignment 2 (Due on 11/18 14:10)

In assignment 2, you are required to extend KVM and support a new hypercall. You should first modify KVM to expose a new hypercall, then test the hypercall in your virtual machine.

NO cheating policy: Please do not copy other people's code from the Internet for this assignment. If we catch you do so, you will get **zero** automatically for this assignment.

1. Before you start

This assignment builds on the Ubuntu environment for KVM for Armv8 that you created in Assignment 1.

2. Adding a new hypercall to KVM (40%)

Hypercalls are the services that the hypervisor expose to virtual machines, which are analogous to system calls that the OS kernels expose to user space applications. Arm provides a special **hvc** instruction to support hypercalls. A VM that executes the hvc instruction traps directly to the VMM (causing a VM exit).

You are required to modify KVM in Linux v5.4 mainline you cloned in assignment 1 to add a new hypercall called **host_cpuid**.

For the assignment, you should first allocate a new hypercall number, and associate the number to a handler in KVM for the new hypercall.

As we discuss in the lecture, a hypervisor implementation usually adopts the time-sharing approach and schedules virtual machines's CPUs (virtual CPUs, or vCPUs) on physical CPUs (pCPUs). KVM does the same. A given vCPU can be executed on any given pCPU.

The **host_cpuid** hypercall allows the VM to query KVM for the **pCPU** ID of the currently running vCPU.

```
unsigned long host_cpuid(void) {
    //return the current pCPU ID of the caller vCPU
}
```

KVM handles the hypercall in highvisor. One thing to note is, as we discussed in the lecture, VM exits are routed to the lowvisor first. The lowvisor then switches to highvisor. The highvisor does some work to handle the VM exit and eventually enters the lowvisor to execute the VM. You should check out our lecture and read KVM's source code to figure how KVM handles the existing hypercalls, and figure out how you could add a new hypercall to KVM.

TIPS:

1. You should read the code for non-VHE in KVM. VHE is an architectural extension from Arm that supports running Linux in EL2 for performance optimization. Our test environment runs KVM with split-mode virtualization (e.g. KVM runs in both EL1 and EL2) and does not support VHE.

2. You could trace KVM's code (note: this is how the highvisor makes hypercalls to the lowvisor) and understand how the hypercall number and arguments are provided and handled in Arm.
3. ctags/cscope is useful when tracing code base

3. Testing the new hypercall (40%)

After you added the **host_cpuid** hypercall to KVM. Next, you should test the newly added hypercall. Arm's hypercall instruction **hvc** is a privileged instruction. Thus, you cannot execute it from user space. Doing this will cause a trap from the guest user space to the guest kernel, generating an undefined error within the VM. Executing **hvc** in user space will ***not*** trap to the VMM!!

In this part of the assignment, you should first add a new system call **sys_host_cpuid** in your guest kernel to invoke **hvc** and issue the **host_cpuid** hypercall. You are then asked to invoke **sys_host_cpuid** from user space.

The **sys_host_cpuid** system call takes no argument, but returns an unsigned long value you got from the **host_cpuid** hypercall. More specifically, your system call handler should do the following:

```
unsigned long sys_host_cpuid(void) {
    // make the host_cpuid hypercall and return the hypercall return value to
user space
}
```

You should first allocate **436** as the system call number for sys_host_cpuid, and add a handler in the guest Linux kernel for handling the system call.

TIPS:

1. In order to trigger the hypercall, some inline assembly may be needed.
2. To invoke the system call from user program, refer to this man page.
3. This file defines all of the system calls supported by Linux.

4. Report (20%)

You are required to provide a report to include the following items:

- (10%) Describe how your hypercall and system call work. E.g. explain how your code functions.
- Describe the behavior that you observe through invoking the hypercall, and answer the following questions:
    - (5%) What is the returned host CPU ID that you got each time? Does the ID remain the same or does it change sometimes?
    - (5%) What does KVM do that results in the different behaviors (e.g. host CPU ID changing or remaining static) that you observe?
- References

Homework submission

You should prepare the following **3** items, and compress them into a file called **"group_id_hw2.zip"**, E.g. "1_hw2.zip".

1. **kvm.patch**: patch for your KVM

2. **vm.patch**: patch for your guest kernel
3. **report.pdf** : Handwritten Report

group_id is your group ID assigned by the TA. You can upload the assignment directly on NTU Cool.

For *.patch file, use the following command to dump the file diff.

```
# git diff > [filename].patch
```