CMPE283 : Virtualization
Assignment 2: Modifying instruction behavior in KVM

Hoang Thy Vo, 012562615, hoangthy.vo@sjsu.edu
Preethi Thimma Govarthanarajan, 012561406, preethi.thimmagovarthanarajan@sjsu.edu

1. Responsibilities Assignment

Implementation/Research	Hoang Thy Vo	Preethi Thimma Govarthanarajan
Kernel build: workflow, errors fix	✓	
Emulate CPUID leaf function for leaf 0x4FFFFFFF	✓	
Return the total number of exits (all types) in %eax	✓	
Research on asm/atomic.h for atomic functions, linux/types.h for atomic_int and atomic64_int types, and export_symbol method for extern variables	✓	
Determine where to place the measurement code		✓
Return the high 32 bits of the total time spent processing all exits in %ebx		✓
Return the low 32 bits of the total time spent processing all exits in %ecx		✓
Write the test module and verify the outputs		✓

Check the frequency of the exits		✓
Check the number of exits of a full reboot	✓	

2. Steps/Instructions

2.1. Set up the host machine

Configure a Linux machine (both VM based and on real hardware) running Ubuntu with VMX/KVM support.
At this step, we utilized the configurations we had made from Assignment 1.

2.2. Build the host kernel

Here are the steps for building the kernel when we configure a totally new host machine:

- sudo bash
- apt-get install build-essential kernel-package fakeroot libncurses5-dev libssl-dev ccache bison flex libelf-dev
- sudo cp /boot/config-$(uname -r) .config
- make menuconfig (and then just exit and save the default .config, don't change anything)
- make && make modules && make install && make modules_install
- reboot

Everytime we update the kvm code, we simply run these steps:

- sudo bash
- make && make modules && make install && make modules_install
- reboot

2.3. Set up the guest machine

- Install virtual machine manager package for Ubuntu/Linux on host machine: sudo apt-get install virt-manager
- Open the virt-manager and create a guest virtual machine running Linux. It can be either a nested virtual machine on top of a virtual machine or a virtual machine on top of a real hardware. We again select Ubuntu to install for this guest machine.
- Install cpuid package: sudo apt-get install cpuid
- Here is the step to run cpuid with leaf 0x4FFFFFFF and get the register values :
    - modprobe cpuid
    - cpuid -l 0x4FFFFFFF

2.4. KVM update

2.4.1 Changes in vmx.c

All changes in vmx.c is performed in function **vmx_handle_exit** which is the first function that is called during a vmexit. **rdtsc()** function is called immediately after all the variable declaration and is stored in a 64 bit **start_exit_time** variable. The atomic variable **exit_count** (for counting number of exits) is incremented by one using **atomic_inc()** function to find the total count of exits.

The **end_exit_time** variable to measure the processor cycle after exit emulation is read in three conditions inside **vmx_handle_exit()** function.

- *if (vmx->emulation_required)*
    - This condition is used in the case of nested virtualization for exit emulation when the guest state is invalid.
- *if (is_guest_mode(vcpu) && nested_vmx_exit_reflected(vcpu, exit_reason))*
    - This condition is also used in the case of nested virtualization. nested_vmx_exit_reflected returns true if vmx has to exit from L2 to L1 to handle exit, or false if exit has to be handled in L0. It is called only in the case of nested virtualization. This *if* condition calls nested_vmx_reflect_vmexit() function to emulate exit reason.
- *if (exit_reason < kvm_vmx_max_exit_handlers && kvm_vmx_exit_handlers[exit_reason])*
    - This *if* condition is valid for normal virtualization. It calls the exit emulation function specific to the exit reason.

In all the above mentioned functions, we read processor's time stamp counter using rdtsc() and store it in **end_exit_time** after the exit emulation is performed. The difference between **end_exit_time** and **start_exit_time** is calculated and stored in a variable **delta**. This delta is then added to 64 bit atomic variable **exit_processing_time** (for counting the time taken to execute the exits) using **atomic64_add()** function to update the total time spend processing all the exits.

2.4.2 Changes in cpuid.c

As cpuid module is inserted before vmx module during kernel build, two atomic variables **exit_count** (for counting number of exits) and **exit_processing_time** (for counting the time taken to execute the exits) were declared and initialized to 0 in cpuid.c file and exported to be accessed in vmx.c

cpuid.c
atomic_t exit_count = ATOMIC_INIT(0);

EXPORT_SYMBOL(exit_count);
atomic64_t exit_processing_time = ATOMIC64_INIT(0);
EXPORT_SYMBOL(exit_processing_time);
vmx.c
extern atomic_t exit_count;
extern atomic64_t exit_processing_time;

handle_cpuid function in **vmx.c** which is called to emulate exits caused for executing cpuid instruction calls **kvm_emulate_cpuid** function in **cpuid.c**. Thus we examined kvm_emulate_cpuid and all other functions in cpuid.c and decided to make changes to handle cpuid with leaf 0x4FFFFFFF in function **kvm_emulate_cpuid.**

In **kvm_emulate_cpuid**, we read the value stored in register eax using kvm_register_read function and check if its equal to 0x4FFFFFFF. If true, then we read 64 bit atomic variable exit_processing_time and split it and write its high 32 bits to ebx register and low 32 bits to ecx register. We then read atomic variable exit_count and write it to eax register.
Extra logs were printed from cpuid.c for testing purposes.

2.5. Verify the output

Here is an example of how we execute the test:
cpuid -l 0x4FFFFFFF
CPU 0:
  0x4fffffff 0x00: eax=0x0061f376 ebx=0x000000da ecx=0x14530d31 edx=0xb26578c0
CPU 1:
  0x4fffffff 0x00: eax=0x0061f397 ebx=0x000000da ecx=0x146089ea edx=0xb26578c0

During the development process, we logged the exit counts, the 64 bits processing time for all exits whenever exits occur. Then when we run the cpuid instruction, we get the syslog, and we cross-check the returned values in registers and the logged values.

3. Comment on the frequency of the exits

Q: Does the number of exits increase at a stable rate?
A: No, the number of exits does not increase at a stable rate for different operations.

Q: Are there more exits performed during certain VM operations?
A: Yes, a full VM reboot will take more exits than other operations.

Q: Approximately how many exits does a full VM boot entail?
A: According to our experiment, a full VM boot takes approximately 2000000 exits.

4. Reference

[1] https://elixir.bootlin.com/linux/v4.7/source/arch/x86/include/asm/msr.h#L161
[2] https://github.com/shichao-an/linux/blob/v2.6.34/include/linux/types.h#L192
[3] https://lkw.readthedocs.io/en/latest/doc/04_exporting_symbols.html
[4] https://www.cyberciti.biz/faq/linux-cpuid-command-read-cpuid-instruction-on-linux-for-cpu/
[4] Intel® 64 and IA-32 Architectures Software Developer Manuals, Volume 2, INSTRUCTION SET REFERENCE, A-L, Table 3-8. Information Returned by CPUID Instruction