CMPE283 : Virtualization
Assignment 2: Modifying instruction behavior in KVM

Due: 28 Apr 2021 before midnight

In this assignment you will learn how to modify processor instruction behavior inside the KVM hypervisor. This lab assignment is worth up to 25 points and may be done in groups of up to **two people max.** Each team member can receive up to 25 points. It is expected that groups of more than one student will find an equitable way to distribute the work outlined in this assignment.

Prerequisites

- You will need a machine capable of running Linux, with VMX or SVM virtualization features exposed. You may be able to do this inside a VM, or maybe not, depending on your hardware and software configuration. You should likely be using the environment you created for assignment 1.

The Assignment

Your assignment is to modify the CPUID emulation code in KVM to report back additional information when a special CPUID "leaf function" is called.

- For CPUID leaf function %eax=0x4FFFFFFF:
  - Return the total number of exits (all types) in %eax
  - 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
    - %ebx and %ecx return values are measured in processor cycles

At a high level, you will need to perform the following:

- Configure a Linux machine, either VM based or on real hardware. You may use any Linux distribution you wish, but it must support the KVM hypervisor.
- Download and build the Linux kernel source code (see below)
- Modify the kernel code with the assignment functionality:
  - Determine where to place the measurement code
  - Create new CPUID leaf 0x4FFFFFFF
    - Report back information as described above
- Create (or otherwise locate) a user-mode program that performs various CPUID instructions required to test your assignment
- Verify proper output

On or before the due date, commit your code to your github repo and post the github repo's URL to the Canvas assignment page for this assignment. Make sure to include a README.md at the TOP level of the repository that includes answers to the questions below.

Testing

You should make a test program that exercises the functionality in your hypervisor modification. Since CPUID can be called at any privilege level, you can make a simple usermode program to do this. A sample test output might look like:

```
$ ./test_assignment2
CPUID(0x4FFFFFFF), exits=32149, cycles spent in exit=1028748293
```

```
$ ./test_assignment2
CPUID(0x4FFFFFFF), exits=32291, cycles spent in exit=1159390993
```

Note: I will not be running your test code. I have my own test code, and I'll be running that. Thus, there is no need for you to include your test code in your submission.

Building The Kernel

To build the kernel (once you have cloned the Linux git repository), the following sequence of commands can be used (eg, for Ubuntu – other distributions have similar steps but may differ in the installation of the build prerequisites):

sudo bash
apt-get install build-essential kernel-package fakeroot libncurses5-dev libssl-dev ccache bison flex libelf-dev
uname -a (and note down your kernel version, for example "4.15.0-112-generic")
cp /boot/config-4.15.0-112-generic ./.config (substitute your version obtained from the previous step here though)
make oldconfig (and then just use the default for everything, don't change anything – you can do this by holding down enter)
make && make modules && make install && make modules-install (will take a long time the first time)
reboot

Verify that you are using the newer kernel (5.8, etc) after reboot:
uname -a

After changing the code in KVM for the assignment, you can rebuild using the same "make" sequence of commands above (and it should only take a few minutes, not several hours).

Special Note Regarding Concurrency

It is possible for a multi-VCPU VM to be handling exits concurrently on more than one CPU (for example, consider a multithreaded program wherein each thread is calling CPUID repeatedly in a loop). You must take care to ensure that the counter value you are keeping inside your KVM code properly handles multiple kernel threads incrementing the same variable simultaneously. (Hint: Look for "atomic variables").

Grading

This assignment will be graded and points awarded based on the following:

- 20 points for the implementation and code producing the output above
- 5 points for the answers to the questions below

Submissions shall be made via pushing code to your github repo and posting the repo's URL to Canvas (like assignment 1). DO NOT WAIT UNTIL LATE ON THE DUE DATE, as server lags or delays may result in a late submission. Since you have four weeks to complete this assignment, I will not accept "server outage or delay" as an excuse for late submissions. If you are concerned about this, submit your assignment early. This is one area that I am extremely picky with – even 1 second late will result in a zero score.

I will be comparing all submissions to ensure no collaboration has taken place. Make sure you do not copy another group's work. If you copy another group's work, members of both groups will receive an F in the class and be reported to the department chair for disciplinary action. If you are working in a group, make sure your partners do not copy another group's work without your knowledge, as all group members will be penalized if cheating is found.

Questions

1. For each member in your team, provide 1 paragraph detailing what parts of the lab that member

implemented / researched. (You may skip this question if you are doing the lab by yourself).

2. Describe in detail the steps you used to complete the assignment. Consider your reader to be someone skilled in software development but otherwise unfamiliar with the assignment. Good answers to this question will be recipes that someone can follow to reproduce your development steps.

Note: I may decide to follow these instructions for random assignments, so you should make sure they are accurate.

3. Comment on the frequency of exits – does the number of exits increase at a stable rate? Or are there more exits performed during certain VM operations? Approximately how many exits does a full VM boot entail?