CMPE-283 Assignment-2 Report

Modify Linux kernel to emulate CPUID(0x4FFFFFFF)

Assignment Purpose

In this assignment we need to modify the linux kernel on KVM to emulate the CPUID(0x4FFFFFFF) instruction. We do so to learn how to modify processor instruction behavior inside the KVM hypervisor.

Detail Recipe:

1) Setup the Linux machine which supports KVM hypervisor

a) Build up on top of google cloud Compute Engine. Check gcloud manual for how to setup a VM.
b) OS version: Ubuntu 18.04.
c) Hardware Resource:
   - CPU: 1 vCPU
   - Memory: 3.75GB
   - Storage: 100G
   - Network: allow https + http + vncserver
d) Enable nested VM:
   - Ref: https://cloud.google.com/compute/docs/instances/enable-nested-virtualization-vm-instances
   - Why is this necessary for assignment 2?
   - Because we need the guest VM to run inside this host VM, enabling nested VM could improve the performance and eliminate the warning from virt-manager on qemu-kvm.
   - Verify the nested VM has been enabled, the following cmd should return 1.

```
# grep -cw vmx /proc/cpuinfo
```

e) Setup the SSH/SCP via gcloud sdk to enable remote access.
f) Setup VNC server to enable remote GUI access
   - Add vnc-server tag in the VM
   - Let the vnc-server tag with port 5901 pass the firewall

2) Clone the linux source code and compile it before touch anything

a) Check the current linux kernel version

```
# uname -a
```

b) Clone the Linux repo:

```
# git clone https://github.com/torvalds/linux.git
# cd linux
```

c) Get information of what Linux commit we are working on, this information is necessary to submit the assignment.

```
# git log
commit 618d919cae2fcaadc752f27ddac8b939da8b441a (HEAD -> master, origin/master, origin/HEAD)
Merge: 5512320c9f6f 2170a0d53bee
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Mon Apr 15 16:48:51 2019 -0700
```

d) Enter root mode:

```
# sudo bash
```

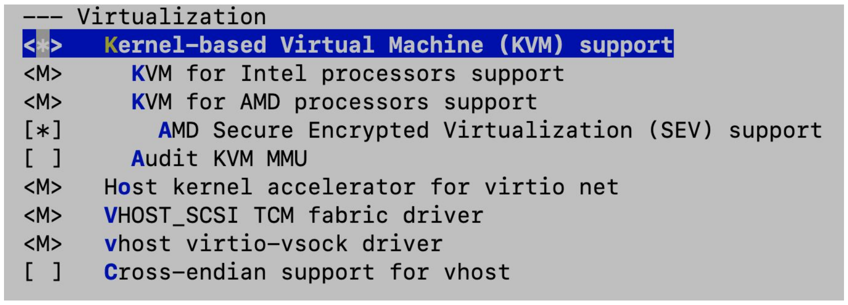
e) Install the build essentials to prepare the compile environment:

```
# apt-get install build-essential kernel-package fakeroot libncurses5-dev libssl-dev ccache bison flex libelf-dev
```

f) Setup the config file:

```
# make menuconfig
```

g) Do the following modification for steps afterwards



```
--- Virtualization
<*>   Kernel-based Virtual Machine (KVM) support
<M>     KVM for Intel processors support
<M>     KVM for AMD processors support
[*]       AMD Secure Encrypted Virtualization (SEV) support
[ ]     Audit KVM MMU
<M>   Host kernel accelerator for virtio net
<M>   VHOST_SCSI TCM fabric driver
<M>   vhost virtio-vsock driver
[ ]   Cross-endian support for vhost
```

h) Compile and build

```
# make && make modules && make install && make modules_install
```

i) One trick for the above step, increase vCPU number and use `make -j#` to accelerate the compile, then revert to single vCPU to save budget.

```
# make -j8 && make modules -j8 && make install -j8 && make modules_install -j8
```

j) reboot, system will automatically select the kernel version we newly built. Verify it:

```
# uname -a
Linux ubuntu18-nested 5.1.0-rc5+ #4 SMP Thu Apr 18 16:09:21 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

3) Setup the nested VM

a) The service behind is qemu-kvm, we are using the GUI wrapper virt-manager to avoid the complex command line arguments. Detailed guide can be found here.

b) Install it:

```
# sudo apt-get install virt-manager
# sudo apt-get install libvirt-bin libvirt-doc
# sudo apt-get install qemu-system
```

c) Launch it with root permission, otherwise can't start the service.

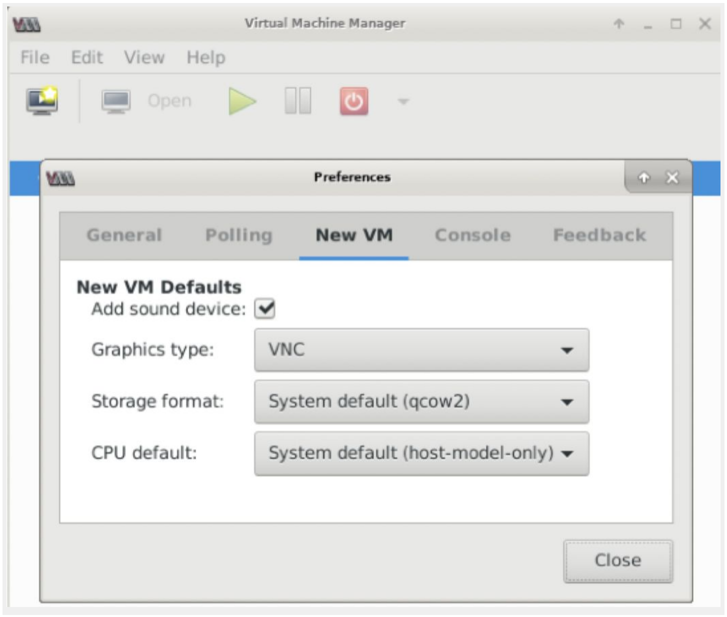
```
# sudo virt-manager
```

d) Prepare the inner VM image, e.g. Ubuntu 18.04.

e) Must enable nested VM, otherwise warnings, performance issue.

f) Start a new VM with the image, following all the prompts to set up the VM. With 1 vCPU, 2G memory, 15G disk.

g) Following all the prompts to install the ubuntu, note that must use the VNC graphics, otherwise can't recognize the keyboard.



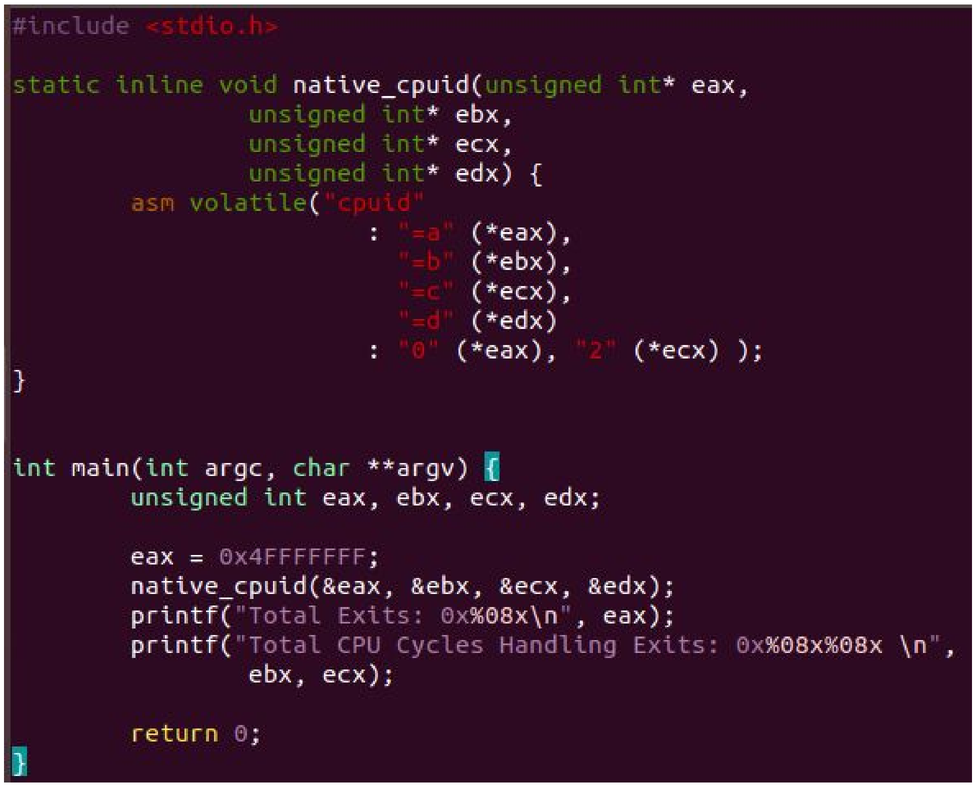
New VM Defaults
Add sound device: ✓
Graphics type: VNC
Storage format: System default (qcow2)
CPU default: System default (host-model-only)

h) Once installation is done, we can write our test code inside the L2 VM, only in this way can our code invoke the KVM modification we made on the L1 VM.

i) Install open-ssh so that we can access the L2 VM via L1 VM.

```
# sudo apt-get install net-tools openssh-server
# ps -aef | grep sshd
```

j) The test code is simple, just invoke the CPUID instruction with the specific argument 0x4FFFFFFF. As shown below.



```
#include <stdio.h>

static inline void native_cpuid(unsigned int* eax,
                                unsigned int* ebx,
                                unsigned int* ecx,
                                unsigned int* edx) {
        asm volatile("cpuid"
                        : "=a" (*eax),
                          "=b" (*ebx),
                          "=c" (*ecx),
                          "=d" (*edx)
                        : "0" (*eax), "2" (*ecx) );
}


int main(int argc, char **argv) {
        unsigned int eax, ebx, ecx, edx;

        eax = 0x4FFFFFFF;
        native_cpuid(&eax, &ebx, &ecx, &edx);
        printf("Total Exits: 0x%08x\n", eax);
        printf("Total CPU Cycles Handling Exits: 0x%08x%08x \n",
                ebx, ecx);

        return 0;
}
```

4) Modify the kernel source code to count exits times and cycles:

a) To count the number of exits, we define an atomic_t variable and initialize it as zero in cpuid.c.
b) The reason for atomic is to ensure thread safety. All the operations should be atomic operations.
c) To use this variable at vmx.c, we need to export it from cpuid.c . And declare in vmx.c as extern.
d) The same logic applies to nu_cycles, except that it should be defined as atomic_long_t, which is 64 bit long.
e) The cpuid.c goes to vmx.ko by default, we need to modify the make configuration to make vmx into the kernel, so that vmx-intel.ko (the destination of vmx.c) could reference it.
f) Source code regarding atomic variable:

```
//definition of atomic_t variable, initialization and export, at cpuid.c
atomic_t num_exits = ATOMIC_INIT(0);
atomic_long_t num_cycles = ATOMIC_INIT(0);
EXPORT_SYMBOL(num_exits);
EXPORT_SYMBOL(num_cycles);

//declare at vmx.c to use them
extern atomic_t num_exits;
extern atomic_long_t num_cycles;

//update the variables with atomic operation, vmx.c
atomic_fetch_add(1, &num_exits);
atomic64_fetch_add((stop_cycle - start_cycle), &num_cycles);

//read the variables with atomic operation, cpuid.c
if (function == 0x4FFFFFFF) {
   *eax = (u32) atomic_read(&num_exits);
   *edx = 0;
   unsigned long tmp = (u64) atomic_long_read(&num_cycles);
   *ecx = (u32) (0xFFFFFFFFLL & tmp);//low 32 bit
   *ebx = (u32) ((0xFFFFFFFF00000000LL & tmp) >> 32); // high 32 bit
   return entry_found;
}
```

g) To count the cycles spent during VMM exit, we are using the RDTSC instruction.

```
//at vmx.c
static uint64_t __rdtsc(void){
   unsigned int lo,hi;
   __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
   return ((uint64_t)hi << 32) | lo;
}
```

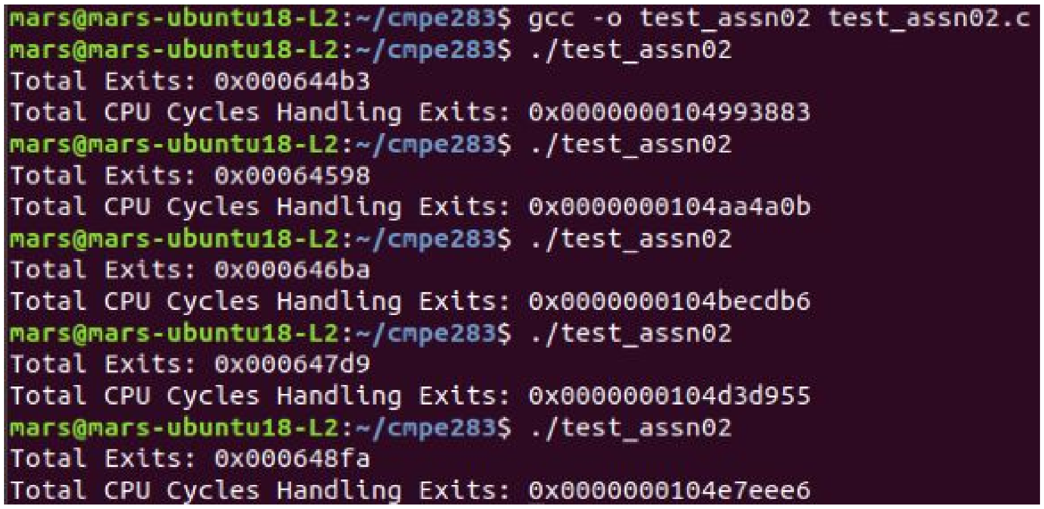
h) Since there are multiple return points of the vmx_handle_exit(), we should count the starting cycles at the entrance and the stopping cycles at all the return points.

```
//at vmx.c
unsigned long long start_cycle, stop_cycle;
start_cycle = __rdtsc();

stop_cycle = __rdtsc();
atomic64_fetch_add((stop_cycle - start_cycle), &num_cycles);
```

5) Test the modification

a) Rebuild the kernel after modifying the source code
b) Reboot the system to enable the new kernel
c) Inside the L1 VM, boot the L2 VM and compile the test code
d) Launch the binary code and check outputs. Both the num_exits and num_cycles should increase as we execute and execute again.


```
mars@mars-ubuntu18-L2:~/cmpe283$ gcc -o test_assn02 test_assn02.c
mars@mars-ubuntu18-L2:~/cmpe283$ ./test_assn02
Total Exits: 0x000644b3
Total CPU Cycles Handling Exits: 0x0000000104993883
mars@mars-ubuntu18-L2:~/cmpe283$ ./test_assn02
Total Exits: 0x00064598
Total CPU Cycles Handling Exits: 0x0000000104aa4a0b
mars@mars-ubuntu18-L2:~/cmpe283$ ./test_assn02
Total Exits: 0x000646ba
Total CPU Cycles Handling Exits: 0x0000000104becdb6
mars@mars-ubuntu18-L2:~/cmpe283$ ./test_assn02
Total Exits: 0x000647d9
Total CPU Cycles Handling Exits: 0x0000000104d3d955
mars@mars-ubuntu18-L2:~/cmpe283$ ./test_assn02
Total Exits: 0x000648fa
Total CPU Cycles Handling Exits: 0x0000000104e7eee6
```

6) Commit modification and generate diff file to submit

a) Git add files to reflect modification in our work directory to the index

```
# git add arch/x86/kvm/vmx/vmx.c
# git add arch/x86/kvm/vmx/vmx.c
```

b) Git commit to stage the modification into our local repository

```
# git commit -m "Added atomic operation: CPE283_Assn02"
```

c) Git log to identify the two commit id which will be used by git diff.

```
# git log
```

commit df7847f45355a63748d6537fdc6671e2ecf77fe7 (**HEAD -> master**)
Author: Mars <mengshi.li@sjsu.edu>
Date: Thu Apr 18 18:58:28 2019 +0000
…
commit 618d919cae2fcaadc752f27ddac8b939da8b441a (**origin/master**, **origin/HEAD**)
Merge: 5512320c9f6f 2170a0d53bee
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date: Mon Apr 15 16:48:51 2019 -0700

d) Git diff between the latest commit with the fresh starting point

```
# git diff <old commit id> <new commit id> > cmpe283-2.diff
```

e) Scp to local OS:

```
# gcloud compute scp
ubuntu18-nested:/home/mengshi_li/linux/linux/cmpe283-2.diff .
```

7) Check the diff file validity

a) Git clone from same repository

```
# git clone https://github.com/torvalds/linux.git
# cd linux
```

b) Check still on the master branch

```
# git branch
```

c) Revert back to the old commid

```
# git reset --hard <commit-id of our starting point>
```

d) Apply the diff file

```
# git apply cmpe283-2.diff
```

e) Config the menu as we required, save it as .config file

```
# make menuconfig
```

f) Compile, build and test in the L2 VM as detailed in step 5).

Experiment Observation:

1) Does the number of exits increase at a stable rate? Or are there more exits performed during certain VM operations?

ANS:
It doesn't increase at stable rate, certain VM operations such as open a file or play video from the disk or download from Internet will trigger more VM exits due to Page Fault Exits and Device IO exits.

2) Approximately how many exits does a full VM boot entail?

ANS:
Records the num_exits <0x5949a> before reboot VM, record num_exits <0x9132a> again once we enter the VM GUI, the difference would be the total exits for a full VM boot. It is approximately 229,000 on my setup. Note that this is very inaccurate counting because we have included the shutdown period.

Reference:

[1] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/p0124r3.html
[2]
https://elixir.bootlin.com/linux/v5.1-rc5/source/include/asm-generic/atomic-instrumented.h#L919
[3] https://lwn.net/Articles/695257/
[4] https://www.kernel.org/doc/html/v4.12/core-api/atomic_ops.html