

實驗目的

In this project, you are going to experience how to build a hosted VM hypervisor and boot up guest virtual machines. You can observe how the hypervisor works and how the traps to VMM are generated by instrumenting the hypervisor code.

實驗器材和所需軟體

- PC x 1
 - Requirement: any x86-based PC.
 - Any linux distribution (Ubuntu is recommended)

Team rule and Due date

- 1) You may do this assignment in a team of two members, however, you are welcome to do it by yourself.
- 2) The source code(VM_HW.tar.bz2) is in <https://goo.gl/jVaVNo>

What to Submit?

- 1) Your **modified** source code: create a directory named *kernel_code* and place your linux kernel code (*only you modified*) in it.
- 2) (Optional) Your own test program to run on the guest system: create a directory named *test_program* and place programs you tested in there.
- 3) Report: In the report, you need to describe and explain where you instrument code to count traps, and discuss your observations of the trap counter behavior. Finally, like a general research report, you need to discuss your experimental results and propose what may be changed in the VM implementation and in experiments for future studies.

All above should be wrapped up in a tar file named *vm_hw2_YourStudentId.tar.gz* and upload to ceiba 作業.

Part I. Setup the experiment environment

Step 0. Install and set Xming and xterm

1. Install and set Xming

Please consider the web <http://blog.jangmt.com/2009/11/xming.html>

2. Install xterm

```
sudo apt-get install xterm
```

Step 1. Build and Boot the Host Linux System on an ARMv8 Simulator

1. Install a cross compiler for ARMv8

```
$ sudo apt-get install gcc-aarch64-linux-gnu
$ aarch64-linux-gnu-gcc // should shows no input files
```

2. Get the files for this homework

```
$ tar zxvf VM_HW2.tar.gz
$ cd VM_HW2
```

- *DDI0487A_a_armv8_arm.pdf*: ARMv8 architecture manual
- *boot_virtio.sh*: Script for booting host system on simulator
- *boot-wrapper*: Bootloader directory.
- *Foundation_Platformpkg*: ARMv8 simulator directory.
- *host.img*: Disk image for host system.
- ***linux-kvm-arm***: ARM Host OS (including KVM hypervisor) source code, *you need to modify this!*

3. Build host OS

```
$ cd linux-kvm-arm
$ ./build.sh // should generate Image in arch/arm64/boot
```

4. Build booting image (=bootloader + OS image for booting ARM host system)

```
$ cd boot-wrapper
$ ./build_virtio.sh // should generate image-foundation_VIRTIO.axf
```

5. Boot ARM host system on simulator

```
$ ./boot_virtio.sh
```

```

FVP terminal_0
Sun May 26 16:51:00 UTC 2013
hwclock: can't open '/dev/misc/rtc': No such file or directory
INIT: Entering runlevel: 5
Starting OpenBSD Secure Shell server: sshd
done.
hwclock: can't open '/dev/misc/rtc': No such file or directory
creating NFS state directory: done
starting 8 nfsd kernel threads: rpc.nfsd: Unable to access /proc/fs/nfsd errno 2
(No such file or directory).
Please try, as root, 'mount -t nfsd nfsd /proc/fs/nfsd' and then restart rpc.nfsd
to correct the problem
done
starting mountd: done
starting statd: done
Starting syslogd/klogd: done
openvt: can't open '/dev/tty1': No such file or directory
Starting auto-serial-console: done
Stopping Bootlog daemon:
bootlogd.
INIT: no more processes left in this runlevel
root@genericarmv8:~# uname -a
Linux genericarmv8 3.14.0 #6 SMP PREEMPT Wed Apr 22 14:02:38 CST 2015 aarch64 GNU
U/Linux
root@genericarmv8:~#

```

6. Access host disk image (If you want to add/get anything to/from ARM host disk)

```

$ mkdir mnt (if there does not exist mnt directory)
$ sudo mount -o loop host.img mnt
$ cd mnt // what you see in mnt is what you will see in host OS's "/" directory
on simulator

```

- You can create file under mnt (with sudo), then the file you created can be access by ARM host.
- The files under mnt/home/root is what you will need to boot guest system.
- After creating files under mnt, you need to umount it.

```

$ cd VM_HW2/
$ sudo umount mnt

```

Step 2. Boot the Guest Linux System on ARMv8 Simulator

This step, all the instruction is executed on ARMv8 simulator rather than PC.

1. Disk content on host system

```

$ cd /home/root
$ ls

```

- *boot_guest.sh*: Script to boot guest OS.
- *screen-4.0.2*: Program to create multiple command console, see next step.
- *image*: Kernel image of guest OS.
- *guest.img*: Disk image for guest OS.
- *lkvm-static*: Program to call KVM and create guest OS, same as the role of *Qemu* described in Part II.

2. Open multiple command console on ARM host

```

$ ./screen-4.0.2

```

- After running screen-4.0.2, you can press CTRL+A-C to create another command console.
- Use CTRL+A-[1~9] to select which command console you want to use.
- Search “linux screen usage” to understand how to use screen.

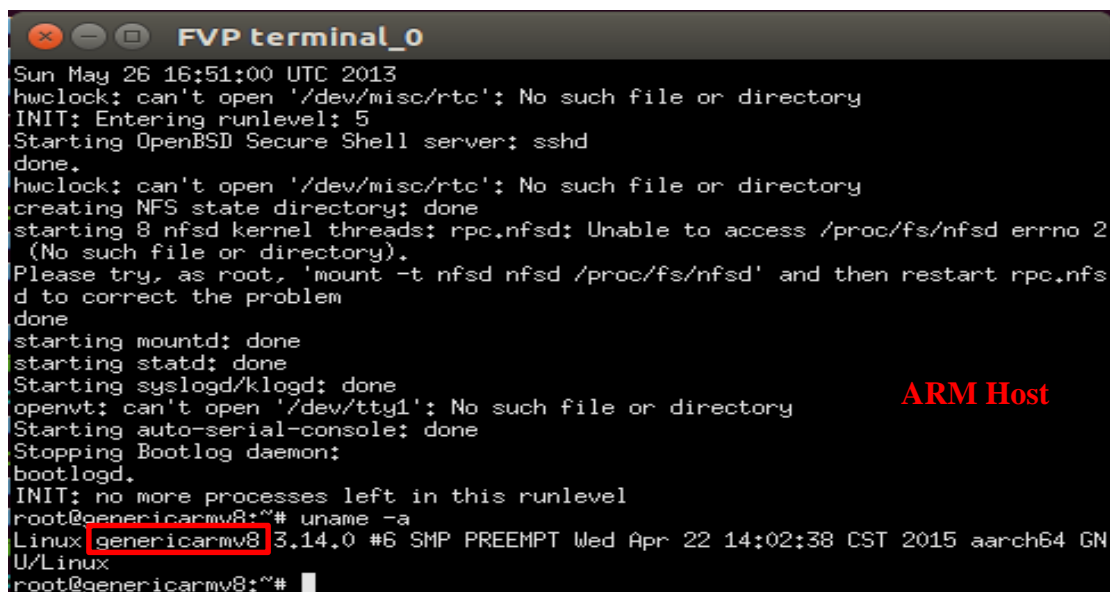
3. Enable Linux tracepoint to dump guest execution information

```
$ echo 1 > /sys/kernel/debug/tracing/event/kvm/kvm_vm_hw2/enable
```

4. Boot guest system

```
$ ./boot_guest.sh
```

- Remember to use screen to create multiple command console before booting guest because guest system will occupy your original command console.

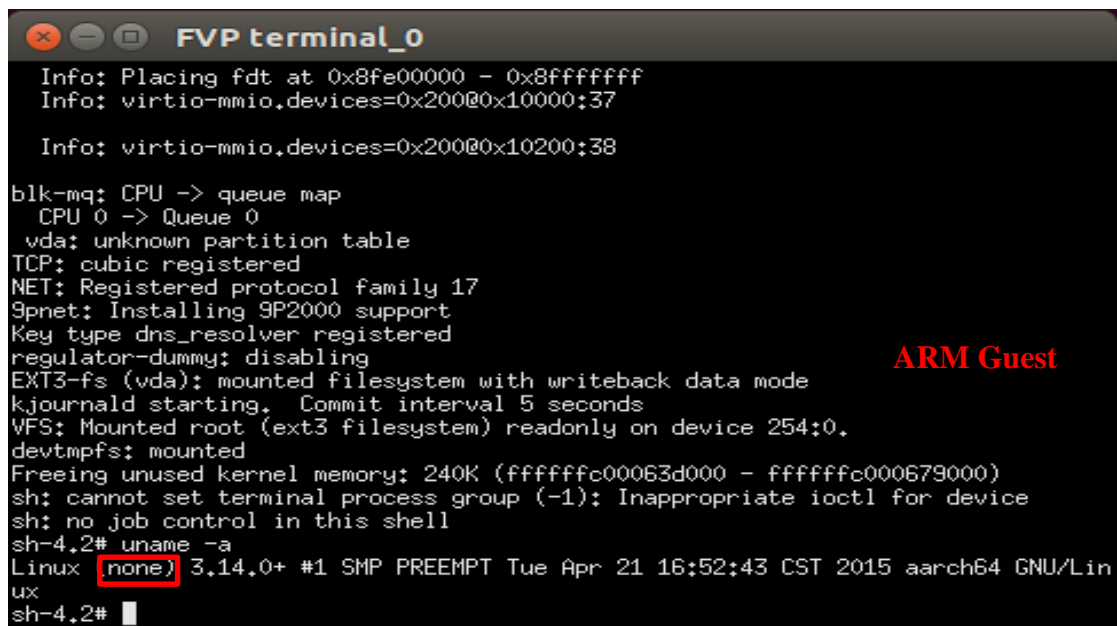


```

FVP terminal_0
Sun May 26 16:51:00 UTC 2013
hwclock: can't open '/dev/misc/rtc': No such file or directory
INIT: Entering runlevel: 5
Starting OpenBSD Secure Shell server: sshd
done.
hwclock: can't open '/dev/misc/rtc': No such file or directory
creating NFS state directory: done
starting 8 nfsd kernel threads: rpc.nfsd: Unable to access /proc/fs/nfsd errno 2
(No such file or directory).
Please try, as root, 'mount -t nfsd nfsd /proc/fs/nfsd' and then restart rpc.nfs
d to correct the problem
done
starting mountd: done
starting statd: done
Starting syslogd/klogd: done
openvt: can't open '/dev/tty1': No such file or directory
Starting auto-serial-console: done
Stopping Bootlog daemon:
bootlogd.
INIT: no more processes left in this runlevel
root@genericarmv8:~# uname -a
Linux genericarmv8 3.14.0 #6 SMP PREEMPT Wed Apr 22 14:02:38 CST 2015 aarch64 GN
U/Linux
root@genericarmv8:~#

```

ARM Host



```

FVP terminal_0
Info: Placing fdt at 0x8fe00000 - 0x8fffffff
Info: virtio-mmio.devices=0x200@0x10000:37
Info: virtio-mmio.devices=0x200@0x10200:38

blk-mq: CPU -> queue map
CPU 0 -> Queue 0
vda: unknown partition table
TCP: cubic registered
NET: Registered protocol family 17
9pnet: Installing 9P2000 support
Key type dns_resolver registered
regulator-dummy: disabling
EXT3-fs (vda): mounted filesystem with writeback data mode
kjournald starting. Commit interval 5 seconds
VFS: Mounted root (ext3 filesystem) readonly on device 254:0.
devtmpfs: mounted
Freeing unused kernel memory: 240K (fffffc00063d000 - ffffffc000679000)
sh: cannot set terminal process group (-1): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.2# uname -a
Linux none 3.14.0+ #1 SMP PREEMPT Tue Apr 21 16:52:43 CST 2015 aarch64 GNU/Lin
ux
sh-4.2#

```

ARM Guest

5. Get guest system execution information

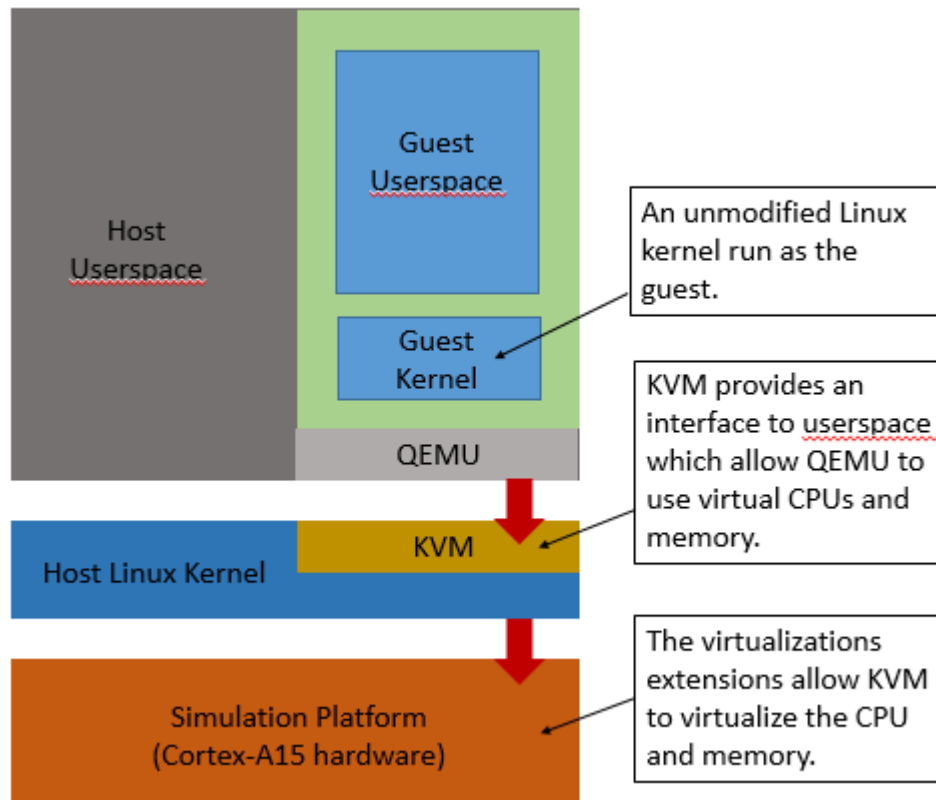
Use screen to change to a host command console.

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

- *For now, you will get CPU, MEM, IO trap number equals to 0 in the trace file.*
- *After you modify the hypervisor source code to count the number of trap, the value of trap should not be 0.*

Part II. KVM/ARM Information

KVM/ARM architecture overview



KVM/ARM allows full virtualization with ARM cortex-a15 virtualization hardware assist.

- CPU virtualization
Hyp mode: hyp mode is more privileged than existing kernel mode. Explicitly separate the virtual machine and hypervisor and have more registers to store important information.
- Memory virtualization
Two stage translation: translate guest virtual to intermediate physical address, then host physical address by MMU.
- IO virtualization
QEMU: emulate most IO operations.
vGIC(virtual generic interrupt controller): allow kernel to program timers when running inside a VM without trapping to the host.

Part III. Profiling the Guest Virtual Machine

This part, all the code you need to modify is under VM_HW2/linux-kvm-arm.

Step 1. Add New Members to VCPU Data Structure

Virtual CPU process the whole guest VM state. When guest VM exit and enter the hypervisor mode, virtual CPU should store the VM state and load the host kernel state.

You need to add members in the VCPU data structure to track how many traps occurred during the guest system execution.

Data structure of virtual CPU

```
(In include/linux/kvm_host.h)  
Struct kvm_vcpu{  
    .....  
};
```

Step 2. Trace How KVM Perform CPU, Memory, IO Virtualization

In virt/kvm/kvm_main: Find the KVM_RUN label.

KVM_RUN is an ioctl command of KVM. You can find the command description in Documentation/virtual/kvm/api.txt. KVM_RUN is sent to KVM each times guest OS is about to run (back from trap) so this is an entry point to trace the how KVM handle guest traps.

In arch/arm/kvm/arm.c: kvm_arch_vcpu_ioctl_run function.

This function is called when KVM get KVM_RUN command and it prepare to do world switch (host switch to guest). Also, this is the function that execute as soon as guest trap and end up calling **handle_exit** which is the **key function to do CPU, memory, I/O virtualization**. Trace the handle_exit function to understand how KVM handle guest faults.

You will need the ARMv8 architecture manual to understand how the type of trap is encoding such that you can identify which trap is caused by cpu fault, memory fault, or I/O fault. Please see the page 1905 in the ARMv8 architecture manual.

Step 3. Instrument Code to Count CPU, Memory, I/O Traps

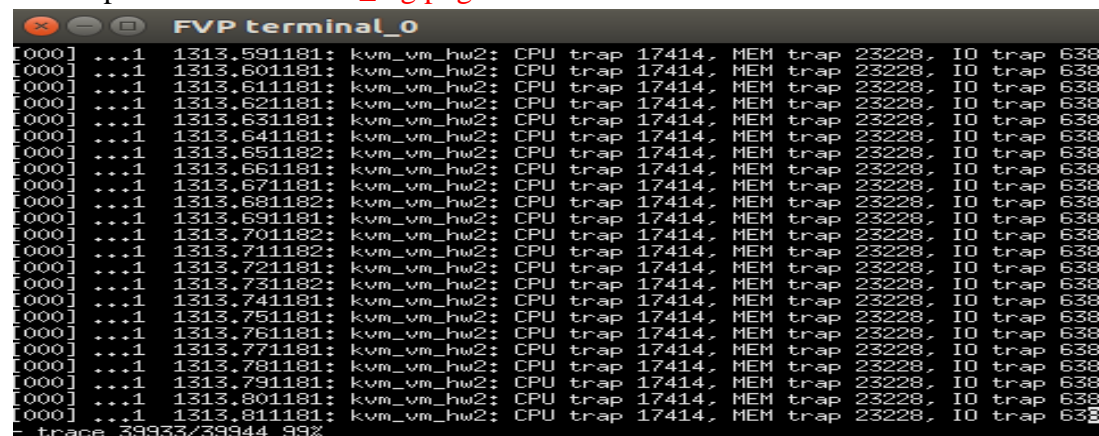
With step2, you should understand how KVM handle all type of fault caused by guest system. **You are asked to add some code to count the number of cpu, memory, I/O trap in any function which you think it is used to handle trap.**

With your instrumentation, the counter should increase during guest system execution. Dump all the trap counter by calling trace_kvm_vm_hw2 which TA has written in the kvm_arch_vcpu_ioctl_run function.

Step 4. Boot Guest and Run Sample Programs

Follow the PartI to boot host and guest system on ARMv8 simulator. As PartI step 2.3, 2.5, enable and dump the tracepoint data after guest system booting.

1. You will get how many trap of different type during system booting. Take a snapshot and save as **boot_log.png**.
2. Run /home/root/mem (*run in the guest OS*) and dump tracepoint data again. Take a snapshot and save as **mem_log.png**.
3. Run /home/root/io (*run in the guest OS*) and dump tracepoint data again. Take a snapshot and save as **io_log.png**.



```
[000] ...1 1313.591181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.601181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.611181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.621181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.631181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.641181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.651182: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.661181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.671181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.681182: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.691181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.701182: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.711182: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.721181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.731182: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.741181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.751181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.761181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.771181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.781181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.791181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.801181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
[000] ...1 1313.811181: kvm_vm_hw2: CPU trap 17414. MEM trap 23228. IO trap 638
- trace 39933/39944 99%
```

For now you will have three png files, boot_log.png, mem_log.png, and io_log.png.

The source of mem, and io is mem.c, and io.c. Trace the code and observe your log, is the trap counter reasonable?

Write a report to describe how many traps you get after guest system booting, mem program execution, io program execution and try to explain it.

You are welcome to write your own test programs to run on the guest system.

Following PartI, step 1.6 to place your test programs onto the guest disk. Remember to use aarch64-linux-gnu-gcc to compile your test programs.