CSIE 5310 Assignment 1 (Due on October 2nd 14:10)

In assignment 1, you are asked to set up the environment, build the software, and run a virtual machine on KVM. Additionally, you will also need to write some KVM code.

The first assignment is based on [KVM for Armv8]. You will first boot the KVM host on an Armv8 hardware emulated by QEMU, then run a virtual machine on the KVM host. Finally, you will dive into the KVM source code and see *CPU virtualization* in action, then use your shiny new environment to test it out.

0. Late submission policy:

- 1 pt deduction for late submissions within a day (before Oct. 3rd 14:10)
- 2 pts deduction for late submissions within 2 days (before Oct. 4th 14:10)
- zero points for submissions delayed by more than 2 days.

1. Before you start

- You should prepare a working Ubuntu environment, which you are allowed to install any software packages as you wished. We recommend installing Ubuntu on a VM (on VMWare workstation/fusion or parallel), or on a bare-metal machine (laptop or lab server) that you have full control. The tutorial provided as follows is based on Ubuntu 22.04 LTS. Please make sure to have at least 50GB free storage in your Ubuntu environment.
- Download the attachment `vm_hw1_files.zip` from NTU Cool into your Ubuntu host. There are 3 files used in this assignment:
    - `run-kvm.sh`
    - `run-guest.sh`
    - `blocker`

2. Running KVM

Compile QEMU

To set up the environment for testing, first clone QEMU from the repo and checkout to version v7.0.0:

```
# git clone https://gitlab.com/qemu-project/qemu.git
# cd qemu/
# git checkout tags/v7.0.0
```

Then configure and compile qemu from the source. You may run into some errors when you do the `configure` command, this is normally because you have missing packages. Google will be your friend for addressing the errors.

```
# cd qemu
# ./configure --target-list=aarch64-softmmu --disable-werror
# make -j4 (-j is to compile in parallel)
# sudo make install
```

Compile Linux/KVM host

Chances are that you are running Ubuntu on an x86 machine, so you will need a cross-compiler to compile the KVM binaries for your Arm based machine. To install the cross-compiler for Arm on your Ubuntu machine, you could `apt install` the `gcc-aarch64-linux-gnu` package.

Once you have QEMU installed, do the following to clone the mainline 5.15 KVM source code.

```
# git clone --depth 1 --branch v5.15 https://github.com/torvalds/linux.git
# cd linux
```

At line 2113 in `arch/arm64/kvm/arm.c` of your Linux kernel source. Put the following to print a message:

```
printk("this is my KVM [Your student ID]\n");
```

Next, compile your KVM host:

```
# make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- defconfig
# make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j4
```

Create virtual disk image

To boot your KVM host, you will need to create a virtual disk image to store its file system.

First, download Ubuntu 20.04's file system binaries here: https://cloud-images.ubuntu.com/releases/focal/release/ubuntu-20.04-server-cloudimg-arm64-root.tar.xz

You can then follow the instructions below to create an Ubuntu 20.04 virtual disk image:

```
# qemu-img create -f raw cloud.img 25g
# mkfs.ext4 cloud.img
# mount cloud.img /mnt
# tar xvf ./ubuntu-20.04-server-cloudimg-arm64-root.tar.xz -C /mnt
# sync
# sudo touch /mnt/etc/cloud/cloud-init.disabled
```

Next, open `/mnt/etc/passwd`, and update the first line to the following to disable root login password.

```
root::0:0:root:/root:/bin/bash
```

Finally, unmount the file system image from your Ubuntu host.

```
# umount /mnt
```

Run KVM host

After the compilation of the Linux kernel source for your KVM host is complete, it is now time to test your newly compiled Linux/KVM binary. You can use the virtual disk image `cloud.img` that you just created above.
We provide you the script `run-kvm.sh` for this.

```
# ./run-kvm.sh -k $PATH_TO_KVM_HOST_IMAGE -i $PATH_TO_YOUR_cloud.img
```

Assuming you put your Linux/KVM source in `/home/ubuntu/`, your `PATH_TO_KVM_HOST_IMAGE` is equal to:

```
/home/ubuntu/arch/arm64/boot/Image
```

Your newly compiled binary for Linux/KVM should be now running. The output you observe is from the virtual serial port.

Configure ssh

You can then login to the machine as a root user without a password. You will have to do the following at the first time when you login to the KVM host to configure `ssh` for your machine:

```
# dpkg-reconfigure openssh-server
```

You need to reconfigure `ssh` to enable root login, modify `/etc/ssh/sshd_config`'s `#PermitRootLogin ..` to `PermitRootLogin yes`. You will also need to set up the `ssh` key authentication for your root user by doing the following on your Ubuntu host:

```
# ssh-keygen
```

Then you should create a new file `/root/.ssh/authorized_keys` in the KVM host, then copy your Ubuntu host's public key (from `~/.ssh/id_rsa.pub`) and paste to `/root/.ssh/authorized_keys`.

Connect to KVM host via ssh

Furthermore, `run-kvm.sh` supports port forwarding from the host (running Ubuntu), so you can `ssh` to your KVM host from your Ubuntu environment. To enable `ssh`, do the following in your VM's (running KVM host) shell:

```
# dhclient
```

Then open another terminal session on your Ubuntu host, and do the following to `ssh` to the KVM host:

```
# ssh root@localhost -p 2222
```

3. Run VM on KVM

The next step is to run a VM on your KVM host.

Compile QEMU to run within KVM

You will need to follow the steps in Compile QEMU to compile QEMU again on your KVM host. You will notice that the QEMU compilation will take a lot more time (30 minutes to an hour, or even longer).

> Note that you **cannot** reuse the QEMU built previously, because chances are that QEMU was built to run on an x86 machine, but now we want to build it to run on an ARM machine.

Create virtual disk image (again)

In the Ubuntu host, follow the steps above to create another virtual disk image, but shrink its size to ~2g for it to fit inside `cloud.img`. Then `scp` the new disk image into KVM host.

```
# scp -P 2222 cloud_inner.img root@localhost:/root
```

Prepare VM kernel image

Simply reuse the kernel image built previously and `scp` it into the KVM host. The kernel image is reusable because it was cross-compiled in the first place.

```
# scp -P 2222 $PATH_TO_LINUX/arch/arm64/boot/Image root@localhost:/root
```

Prepare script

`scp` `run-guest.sh` into KVM host as well:

```
# scp -P 2222 $PATH_TO_FILES/run-guest.sh root@localhost:/root
```

Run VM on KVM

The script `run-guest.sh` is almost the same as `run-kvm.sh`, except that it uses a smaller RAM in the virtual machine and sets the `enable-kvm` flag in QEMU run on KVM. If the flag is not set, QEMU will run the VM using binary translation.

Similarly, you need to specify the path of guest kernel image, and a virtual disk image. The command looks like the following:

```
# ./run-guest.sh -k PATH_TO_KERNEL_IMAGE -i PATH_TO_YOUR_disk.img
```

Finally, you should be able to login to your VM.

4. KVM CPU virtualization

By now you should have set up your environment. Let's explore some KVM internals, regarding the topic of **CPU virtualization** introduced in lecture. **Your task is to patch the KVM host in order to boot the VM running our specially curated executable `blocker` as the init task**.

In other words, you should make the guest VM boot running `blocker` as the init task by **modifying the source code of KVM**, then generate and submit the patch. Modifying QEMU, `blocker`, or the guest kernel image is **NOT** allowed.

Place `blocker` into the guest VM

Firstly, place `blocker` in any path (e.g. `/root/blocker`) of your choosing in the guest VM's disk image (the image used to run the guest VM). You can first `scp` the `blocker` executable into KVM host's file system, then mount the guest VM's image and copy `blocker` into the guest VM's file system.

Assign `blocker` as the init for the guest VM

Then make `blocker` the init task of the VM by modifying `run-guest.sh` like so:

```
# originally
CMDLINE="earlycon=pl011,0x09000000"
# change to (/root/blocker is wherever you place the blocker executable)
CMDLINE="earlycon=pl011,0x09000000 init=/root/blocker"
```

This appends an `init=..` option to the kernel command line, overriding the default of `/sbin/init`, `/etc/init`, `/bin/init`, and `/bin/sh` (located at `init/main.c:1560`).

Now restart your VM, after Linux initializes as it prints the first batch of kernel logs, `blocker` will get executed as the init task, and it will tell you what you should do to continue to boot the system.

5. Homework submission

You should submit the assignment via NTU Cool.

You are required to provide:

- video recording of how you boot KVM then run the VM
- patch for KVM host

Submission format and grading criteria

Environment setup recording (8 pts total)

For the recording, name the video file with `[Student-ID]_hw1.mp4`. For example, if your student number is r01234567, then your file name should be `[Student-ID]_hw1.mp4`.

After your KVM host is booted, you are required to run the following command, which should output some messages like the following. The output shows that KVM has been initialized correctly on Arm hardware with virtualization extensions.

```
# dmesg | grep -i kvm
[    0.311061] kvm [1]: Guests without required CPU erratum workarounds can deadlock system!
[    0.311430] kvm [1]: IPA Size Limit: 44 bits
[    0.315611] kvm [1]: vgic interrupt IRQ9
[    0.317198] kvm [1]: Hyp mode initialized successfully
[    0.317295] this is my KVM [your student ID]
```

Next, you should `ssh` to your KVM host from your Ubuntu host, and execute the script to run the guest VM.

Once you boot the virtual machine, you should do `dmesg` again, which shows a different output. This is because the virtual machine does not include the support of Arm's virtualization extensions, which KVM needed to initialize properly.

```
# dmesg | grep -i kvm
[    0.000000] smccc: KVM: hypervisor services detected (0x00000000 0x00000000 0x00000000 0x00000003)
[    3.150518] kvm [1]: HYP mode not available
```

Please make sure to record the following in your video:

- Both of the `dmesg` in your video and clearly shows the respective results
    - `dmesg` on KVM host (3.5 pts)

    - `dmesg` on guest VM (3.5 pts)
- How you `ssh` to the host KVM from your host Ubuntu to run the VM (1 pt)

KVM patch (2pts total)

For the patch, go to the root of your Linux source, then use

```
# git diff > file_name.patch
```

to generate a patch and name it `[Student-ID]_hw1.patch`. For example, if your student number is r01234567, then your file name should be `r01234567_hw1.patch`.

We will apply the patch you submitted by issuing `git apply [Student-ID]_hw1.patch`, no credit will be given if your patch does not get applied successfully. Please double-check before you submit.

If your patch successfully boots the guest VM running `blocker` as init, 2pts will be given.

Submission to NTU Cool

Please place the video file and the patch into a folder named `[Student-ID]_hw1`. For example, if your student number is r01234567, then your folder name should be `r01234567_hw1`. The folder structure should be the same as follows.

```
[Student-ID]_hw1
|---- [Student-ID]_hw1.mp4
└---- [Student-ID]_hw1.patch
```

Then, compress the folder into `[Student-ID]_hw1.zip` and submit it to NTU Cool.