Assignment 2 -- Build the Linux kernel

Objectives

This exercise instructs you how to build a Linux kernel and run it in a QEMU environment.

Required Setup

Assume you already have a Ubuntu 24.04 server running either in a VirtualBox or a native machine.

First, let's install a few pre-requisite Debian packages that will allow us to compile the Linux kernel.

```
sudo apt update
sudo apt install -y build-essential git libssl-dev flex bison wget pkg-config libelf-dev
sudo apt install -y qemu-system-x86 debootstrap
sudo apt install -y cloud-utils genisoimage
```

Part 1: Build the Linux kernel (~25 minutes)

We will compile a minimal version of the Linux kernel that can run in a QEMU virtual machine. To get started, navigate to the Linux source code directory (assuming the Git repository is located in your home directory) and switch to the `v6.8` branch:

```
cd ~/linux
git checkout v6.8
```

Next, we'll use the default configuration for a basic x86-64 machine. That configuration can be generated as follows:

```
make defconfig
```

The `make defconfig` will use `x86_64_defconfig` on an x86_64 machine. Once done launch the compilation of the kernel with the following command:

```
make -j4
```

This will take a while (10~30 minutes, depending on your machine settings). Once finished, the compiled kernel's binary is `arch/x86/boot/bzImage`.

T1: **4 points** for **building the kernel**. You can take a screenshot and attach it to the assignment report.

Try out the kernel with QEMU (~2 minutes)

Try to boot the kernel we just built with QEMU as follows:

```
qemu-system-x86_64 -m 1G -kernel arch/x86_64/boot/bzImage -nographic -append "console=ttyS0"
```

Here we tell QEMU to create a machine with `1G` of RAM and to use the kernel we compiled. The option `-append` indicates that the option `"console=ttyS0"` should be passed to the kernel, telling it to output its boot log on the VM's serial console.

You will see Linux starting to boot, but the boot process ends with the following error:

```
/dev/root: Can't open blockdev
VFS: Cannot open root device "(null)" or unknown-block(0,0): error -6
Please append a correct "root=" boot option; here are the available partitions:

Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
```

Don't panic! This is because the kernel we built cannot find a root filesystem. The root filesystem contains all user-space utilities, such as shell.

To **exit QEMU**, press `Ctrl+a` followed by `x`.

T2: **2 points** for **seeing this kernel panic and exit QEMU**. You can take a screenshot and attach it to the assignment report.

Part 2: Create and launch a QEMU virtual machine using Ubuntu Cloud Images and Cloud-init (~20 minutes)

Download an Ubuntu Cloud Image

We will use the Ubuntu 24.04 LTS (Noble Numbat) cloud image for this assignment.

```
wget https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-amd64.img
```

Prepare the cloud-init (~10 minutes)

Save the following config into a file named `user-data`. Make sure to replace `<your-ssh-public-key>` with your **SSH public key**.

```
#cloud-config
users:
  - name: ubuntu
    ssh-authorized-keys:
      - <your-ssh-public-key>
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: sudo
    shell: /bin/bash
chpasswd:
  list: |
    ubuntu:ubuntu
  expire: False
```

And save the following into `meta-data`:

```
local-hostname: ubuntu-lkp
```

Generate the `cloud-init.iso`

```
genisoimage -output cloud-init.iso -volid cidata -joliet -rock user-data meta-data
```

Resize the virtual machine disk image:

```
qemu-img resize noble-server-cloudimg-amd64.img +4G
```

You can check the VM image using `qemu-img info noble-server-cloudimg-amd64.img`.

Launch the Ubuntu cloud image (w/o your Linux kernel)

The Ubuntu cloud image contains a Linux kernel. We can first boot it up and let cloud-init to finish the configuration. Copy, paste and execute the following command in the shell:

```
qemu-system-x86_64 \
  -smp 2 -m 2G -nographic \
  -hda ./noble-server-cloudimg-amd64.img \
  -device e1000,netdev=net0 -netdev user,id=net0,hostfwd=tcp::2200-:22 \
  -cdrom ./cloud-init.iso
```

- `-hda ./noble-server-cloudimg-amd64.img` specifies the virtual hard disk file to use as the main disk for the VM.
- `-device e1000,netdev=net0` adds a virtual network device (e1000) connected to the virtual network defined by `-netdev`.
- `-netdev user,id=net0,hostfwd=tcp::2200-:22` configures `user-mode` networking for the VM (net0) and sets up port forwarding from the host's port 2200 to the guest's SSH port (22).
- `-cdrom ./cloud-init.iso` attaches the `cloud-init.iso` file as a virtual CD-ROM for initial configuration (e.g., set up hostname, SSH key, etc.).

It will take about 1 minute to see the virtual machine booted. When the login prompt appears:

... ...

```
[  109.098393] cloud-init[852]: Cloud-init v. 24.3.1-0ubuntu0~24.04.2 running 'modules:c.
[  116.319845] cloud-init[862]: Cloud-init v. 24.3.1-0ubuntu0~24.04.2 running 'modules:final' at Tue,
31 D.
[  116.710797] cloud-init[862]: Cloud-init v. 24.3.1-0ubuntu0~24.04.2 finished at Tue, 31 Dec 2024
23:43:0s

Ubuntu 24.04.1 LTS ubuntu-lkp ttyS0

ubuntu-lkp login:
```

Login as `ubuntu`, and use `ubuntu` as the password.

You can check the kernel version using `uname -a`, and shut down the VM:

```
sudo poweroff
```

You only need to use the `cloud-init.iso` the **first time** you boot the VM to configure the initial settings (e.g., creating users, setting up SSH keys, hostname, etc.). Once the VM is booted and configured, the configuration is saved to the VM's filesystem, and `cloud-init` will not require the ISO for subsequent boots.

Launch the VM with your Linux kernel (~5 minutes)

Save the following script as a file named `run-ubuntu.sh`:

```
#!/bin/bash
QEMU_BIN=qemu-system-x86_64
NCPU=2
MEMSIZE=2G

KNL_SRC=./linux     # TODO: Change with your kernel base location
BZIMAGE=${KNL_SRC}/arch/x86/boot/bzImage
CMDLINE="nokaslr console=ttyS0 root=/dev/sda1 rw debug"

UBUNTU_IMG=noble-server-cloudimg-amd64.img

sudo ${QEMU_BIN} \
    -s \
    -nographic \
    -smp ${NCPU} -m ${MEMSIZE} \
    -nic user,host=10.0.2.10,hostfwd=tcp:127.0.0.1:2200-:22 \
    -net nic,model=e1000 \
    -drive file=${UBUNTU_IMG},format=qcow2 \
    -kernel ${BZIMAGE} \
    -append "${CMDLINE}"
```

Note: If you are **not** running QEMU within a VirtualBox VM or other virtualized environment, you can add `-enable-kvm` to accelerate the VM execution.

```
chmod +x run-ubuntu.sh
./run-ubuntu.sh
```

Once the VM is booted with your compiled kernel, you should be able to `ssh` into this VM from another terminal using:

```
ssh ubuntu@localhost -p 2200
Check the Linux kernel version:
ubuntu@ubuntu-lkp:~$ uname -a
Linux ubuntu-lkp 6.8.0 #3 SMP PREEMPT_DYNAMIC Tue Dec 31 05:15:29 UTC 2024 x86_64 x86_64 x86_64
GNU/Linux
```

T3: **4 points** for **seeing the new kernel version**. You can take a screenshot and attach it to the assignment report.