Brenda Wang
COEN 241 Cloud Computing
1 February 2023
HW1

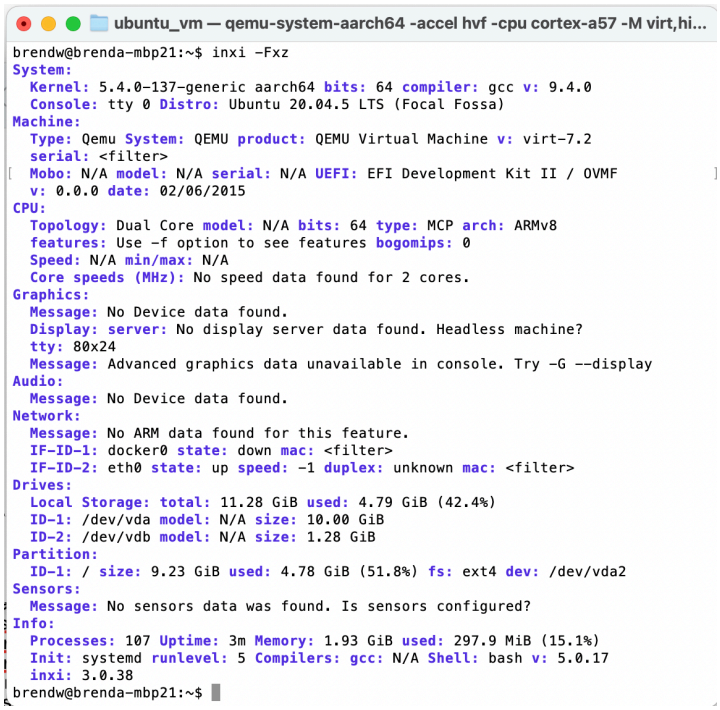
1. Configurations

Host System configuration

MacBook Pro
MacOS Monterey
Chip: Apple M1 Pro (10-core CPU, 32-core GPU)
Memory: 16 GB
Storage: 1 TB Flash Storage

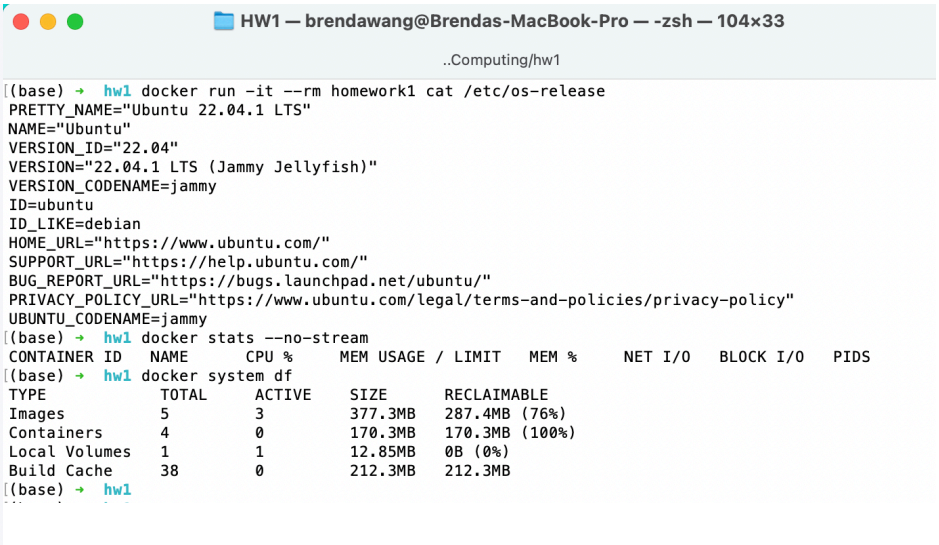
QEMU configuration

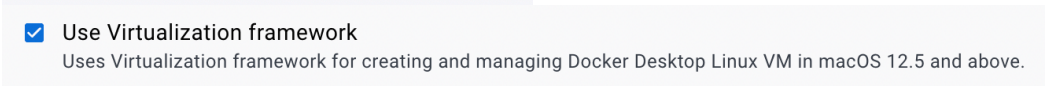
QEMU Virtual Machine
Ubuntu 20.04.5 LTS
CPU architecture: ARMv8
Memory: 2 GB
Storage: 10 GB



Docker configuration

Ubuntu 22.04.1 LTS
CPUs: 5
Memory: 8 GB
Storage: 64 GB





2. Installation and Experiment Commands

QEMU

- Install QEMU using homebrew and confirm

```
$ brew install qemu
$ qemu-system-aarch --version
     > QEMU emulator version 7.2.0
```

- Create a qcow2 (QEMU image format copy on write) virtual disk for the image

```
$ qemu-img create -f qcow2 ubuntu_drive.qcow2 10G
```

-

```
$ dd if=/dev/zero conv=sync bs=1m count=64 of=ovmf_vars.fd
```

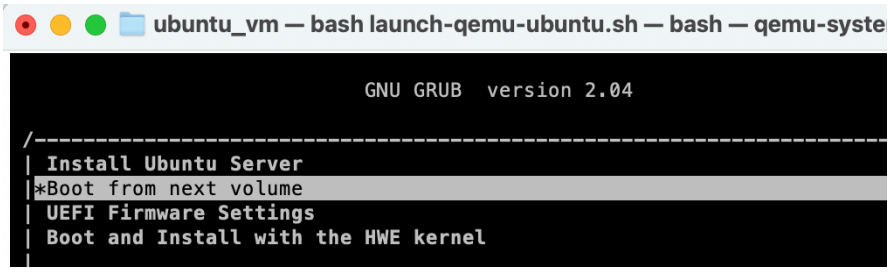
- Download the Ubuntu ISO file

https://cdimage.ubuntu.com/releases/20.04/release/ubuntu-20.04.5-live-server-arm64.iso

- Install the Ubuntu VM by following the instructions in the Installer

```
$ qemu-system-aarch64
-accel hvf
-cpu cortex-a57
-M virt,highmem=off
-m 2048
-smp 2
-nographic
-drive "file=/opt/homebrew/share/qemu/edk2-aarch64-
code.fd,if=pflash,format=raw,readonly=on"
-drive "if=none,file=ubuntu_drive.qcow2,format=qcow2,id=hd0"
-drive "file=/Users/brendawang/Documents/SCU/SCU-w23/COEN241-Cloud-
Computing/ubuntu_vm/ovmf_vars.fd,if=pflash,format=raw"
-device virtio-blk-device,drive=hd0,serial="dummyserial"
-device virtio-net-device,netdev=net0
-cdrom "/Users/brendawang/Documents/SCU/SCU-w23/COEN241-Cloud-Computing/
ubuntu_vm/ubuntu-20.04.5-live-server-arm64.iso"
-netdev user,id=net0
-vga none
-device ramfb
-device usb-ehci
-device usb-kbd
-device usb-mouse -usb
```

- If the server doesn't automatically reboot after installing, rerun the previous command for future boots (I couldn't figure out the command so I just run the same command and choose "Boot from next volume" in the subsequent GNU Bootloader screen



- Write the shell scripts for the CPU and Fileio tests

```
brendw@brenda-mbp21:~$ sudo apt update
brendw@brenda-mbp21:~$ sudo apt intall sysbench
brendw@brenda-mbp21:~$ vim cpu-tests.sh
brendw@brenda-mbp21:~$ chmod u+x cpu-tests.sh
brendw@brenda-mbp21:~$ vim fileio-tests.sh
brendw@brenda-mbp21:~$ chmod u+x fileio-tests.sh
brendw@brenda-mbp21:~$ bash cpu-tests.sh
brendw@brenda-mbp21:~$ sudo bash fileio-tests.sh
```

- To exit QEMU, type
  > ctrl+A, X

QEMU Options	
accel	hardware accelerator
cpu	processor architecture to emulate
M	machine (necessary to specify)
m	memory
smp	number of cores the guest is permitted to use
nographic	run directly in terminal
drive	disk images, storage devices
device	usb emulation; virtual block devices
CD-ROM	optical drive
netdev	create TCP and UDP connections for connections to VM
vga	graphics card

Docker

- Download Docker Desktop from https://docs.docker.com/desktop/mac/apple-silicon/ and install it

- Check https://hub.docker.com/_/ubuntu for correct version for host machine

```
$ docker pull arm64v8/ubuntu
```

- Confirm Ubuntu image was pulled

```
$ docker images
```

- Run the image (-i interactively, -t with pseudo tty, --rm automatically remove the container after exiting)

```
$ docker run –it ––rm <image–id> bash
```

- Install vim and sysbench to write and run the experiment scripts

```
root@ef6a6f3083cc:/# apt–get update
root@ef6a6f3083cc:/# apt–get install sysbench
root@ef6a6f3083cc:/# apt–get install vim
```

- Write the shell scripts for the CPU and Fileio tests

```
root@ef6a6f3083cc:/# mkdir –p /home/brenda/hw1
root@ef6a6f3083cc:/# cd /home/brenda/hw1
root@ef6a6f3083cc:/home/brenda/hw1# vim cpu–tests.sh
root@ef6a6f3083cc:/home/brenda/hw1# chmod u+x cpu–tests.sh
root@ef6a6f3083cc:/home/brenda/hw1# vim fileio–tests.sh
root@ef6a6f3083cc:/home/brenda/hw1# chmod u+x fileio–tests.sh
root@ef6a6f3083cc:/home/brenda/hw1# bash cpu–tests.sh
root@ef6a6f3083cc:/home/brenda/hw1# bash fileio–tests.sh
```

- In another terminal window, commit the container to a new image and confirm it has been created

```
$ docker ps
$ docker commit <container–id> homework1
$ docker images
```

3. Virtualized Ubuntu SysBench Experiments & Results

CPU Test

** Each request consists in calculation of prime numbers up to a value specified by the --cpu-max-primes option.

Benchmark Command:

sysbench --test=cpu [parameters] run | grep … >> $FILE

Parameters:

Test 1: —cpu-max-prime=10000, (default --max-time=10)

Test 2: --cpu-max-prime=30000, (default --max-time=10)

Test 3: --cpu-max-prime=10000, --max-time=30

Test 4: --cpu-max-prime=30000, --max-time=30

FileIO Test

** Each thread performs specified I/O operations on this set of files to produce various kinds of file I/O workloads. Each data read from disk requires a checksum validation. Each write operation fills the block with random values then the checksum is calculated and stored with the offset. Each read from block requires a validation of the stored offset and checksum with the real ones. It was necessary to purge the cache after each test so that the benchmark speeds would not get faster because the test could access cached files.

Benchmark Commands:

sysbench --num-threads=16 --test=fileio [parameters] --file-test-mode=rndrw prepare &>/dev/null

sysbench --num-threads=16 --test=fileio [parameters] --file-test-mode=rndrw run | grep … >> $FILE

sysbench --num-threads=16 --test=fileio [parameters] —file-test-mode=rndrw cleanup

sync && apt-get purge

Parameters:

Test 1: (default--file-num=128), --total-size=3G

Test 2: --file-num=256, --total-size=3G

Test 3: (default--file-num=128), --total-size=1G

Test 4: --file-num=256, --total-size=1G

Each test is repeated 5 times and the average of each statistic is presented in the following tables.

The result from each `sysbench` test is piped to `grep 'avg:\|min:\|max:\|execution time'` and appended to the results `$FILE` to save the relevant statistical data.

*** from SysBench Manual*

a) CPU test - System Virtualization (QEMU)

	Test 1 Average	Test 2 Average	Test 3 Average	Test 4 Average
MIN	0.09	0.39	0.09	0.39
AVG	0.09	0.40	0.09	0.402
MAX	0.93	0.672	53.48	47.586
STDDEV	0.009009	0.040014	0.003003	0.013406

b) FileIO test - System Virtualization (QEMU)

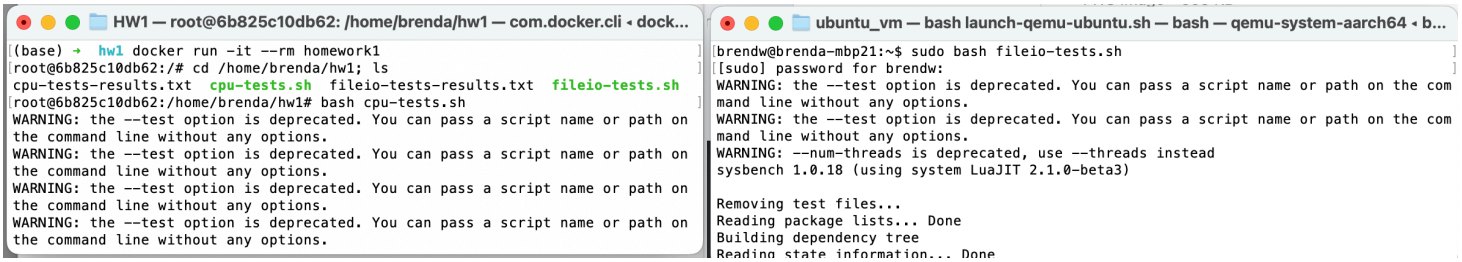
	Test 1 Average	Test 2 Average	Test 3 Average	Test 4 Average
MIN	0.00	0.00	0.00	0.00
AVG	0.202	0.18	0.14	0.128
MAX	29.18	120.934	13.232	43.226
STDDEV	0.020274	0.018085	0.014053	0.012862

c) CPU test - OS Virtualization (Docker)

	Test 1 Average	Test 2 Average	Test 3 Average	Test 4 Average
MIN	0.11	0.458	0.11	0.45
AVG	0.12	0.47	0.12	0.35
MAX	0.678	3.254	0.846	1.728
STDDEV	0.0122019	0.047033	0.004006	0.011770

d) FileIO test - OS Virtualization (Docker)

	Test 1 Average	Test 2 Average	Test 3 Average	Test 4 Average
MIN	0.00	0.00	0.00	0.00
AVG	0.18	0.15	0.184	0.15
MAX	10.784	10.97	10.76	11.538
STDDEV	0.018063	0.015058	0.018463	0.015057

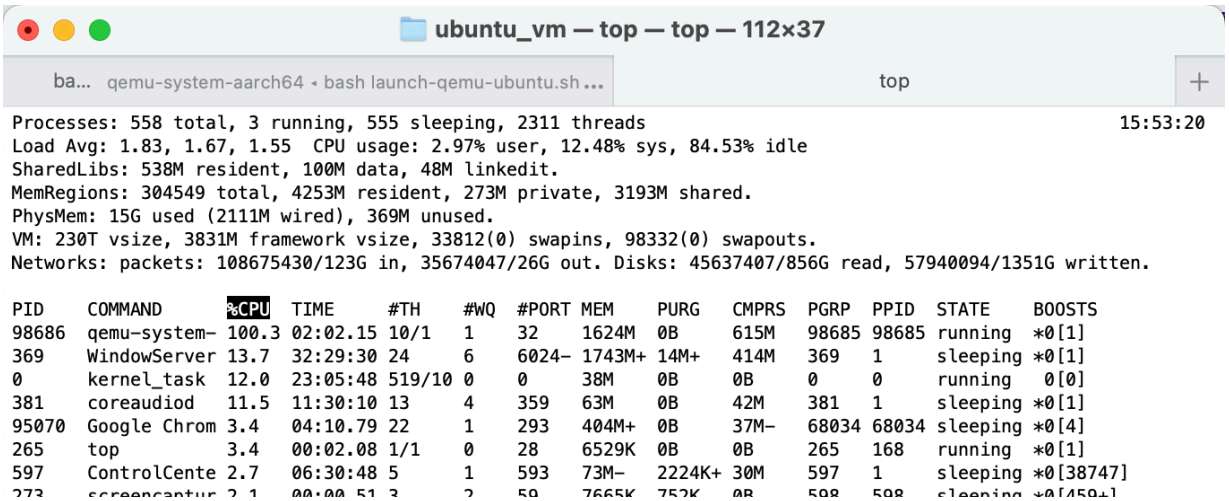


4. Performance

The performance data of the host machine during each test and virtualization was observed using the `top` command. The %CPU is based on the total number of CPUs of the host machine.

> During the CPU test, the CPU utilization was about the same.

QEMU CPU:



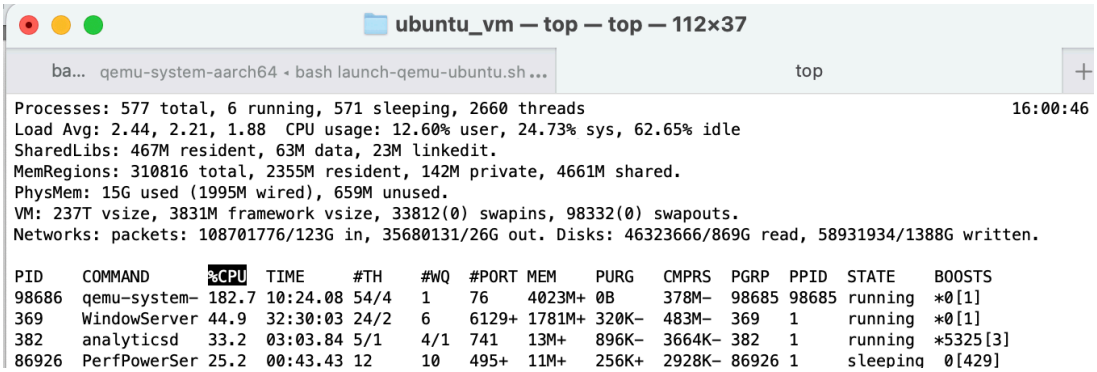

```
Processes: 558 total, 3 running, 555 sleeping, 2311 threads                                15:53:20
Load Avg: 1.83, 1.67, 1.55  CPU usage: 2.97% user, 12.48% sys, 84.53% idle
SharedLibs: 538M resident, 100M data, 48M linkedit.
MemRegions: 304549 total, 4253M resident, 273M private, 3193M shared.
PhysMem: 15G used (2111M wired), 369M unused.
VM: 230T vsize, 3831M framework vsize, 33812(0) swapins, 98332(0) swapouts.
Networks: packets: 108675430/123G in, 35674047/26G out. Disks: 45637407/856G read, 57940094/1351G written.

PID    COMMAND      %CPU  TIME     #TH    #WQ  #PORT MEM    PURG   CMPRS  PGRP  PPID  STATE    BOOSTS
98686  qemu-system- 100.3 02:02.15 10/1   1    32    1624M  0B     615M   98685 98685 running  *0[1]
369    WindowServer 13.7  32:29:30 24     6    6024- 1743M+ 14M+   414M   369   1     sleeping *0[1]
0      kernel_task  12.0  23:05:48 519/10 0    0     38M    0B     0B     0     0     running   0[0]
381    coreaudiod   11.5  11:30:10 13     4    359   63M    0B     42M    381   1     sleeping *0[1]
95070  Google Chrom 3.4   04:10.79 22     1    293   404M+  0B     37M-   68034 68034 sleeping *0[4]
265    top          3.4   00:02.08 1/1    0    28    6529K  0B     0B     265   168   running  *0[1]
597    ControlCente 2.7   06:30:48 5      1    593   73M-   2224K+ 30M    597   1     sleeping *0[38747]
273    screencaptur 2.1   00:00.51 3      2    59    7665K  752K   0B     598   598   sleeping *0[459+]
```

Docker CPU:

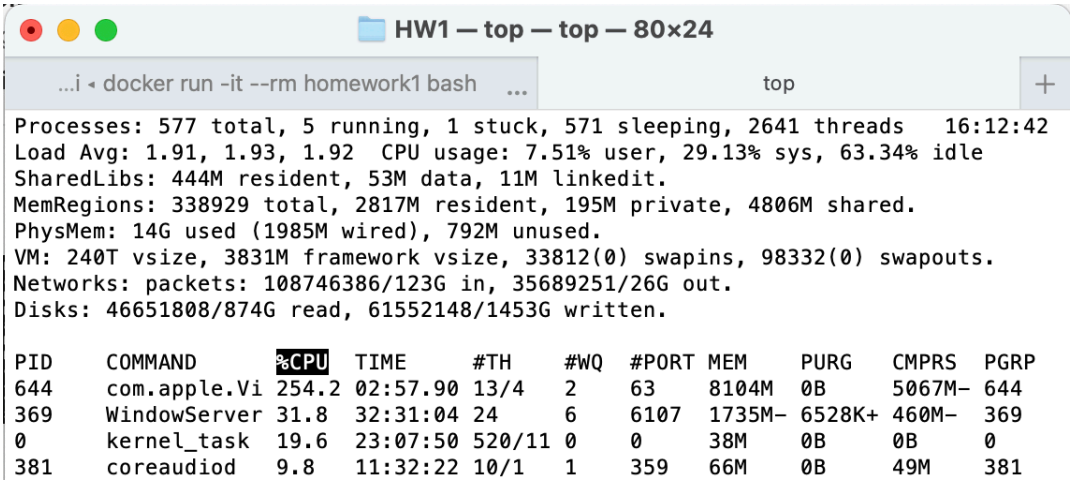

```
Processes: 578 total, 4 running, 574 sleeping, 2617 threads             16:11:15
Load Avg: 1.83, 1.90, 1.91  CPU usage: 2.81% user, 12.75% sys, 84.42% idle
SharedLibs: 446M resident, 53M data, 13M linkedit.
MemRegions: 338997 total, 2776M resident, 192M private, 1646M shared.
PhysMem: 14G used (2003M wired), 1246M unused.
VM: 241T vsize, 3831M framework vsize, 33812(0) swapins, 98332(0) swapouts.
Networks: packets: 108742246/123G in, 35688467/26G out.
Disks: 46634037/874G read, 61145580/1438G written.

PID    COMMAND      %CPU  TIME     #TH    #WQ  #PORT MEM    PURG   CMPRS  PGRP
644    com.apple.Vi 100.7 01:13.09 12/1   1    62    8102M  0B     8424M  644
0      kernel_task  13.1  23:07:40 519/11 0    0     38M    0B     0B     0
381    coreaudiod   11.7  11:32:12 12     3    360   66M    0B     49M    381
369    WindowServer 10.1  32:30:55 23     5    6102- 1736M- 7904K- 475M   369
2931   top          5.6   00:01.91 1/1    0    28    9489K  0B     0B     2931
95070  Google Chrom 3.2   04:48.91 23/1   1    296   408M+  0B     186M   68034
2984   screencaptur 2.4   00:00.41 2      1    60    7874K  752K   0B     598
```

> During the FileIO test, Docker (process: com.apple.Virtualization.VirtualMachine) had higher CPU utilization than QEMU did, which was not expected as containers should be more lightweight.

QEMU FileIO:



Docker FileIO:



5. Git Repository Information

https://github.com/brendw/coen241/tree/main/hw1

6. Extra Credit

Dockerfile

FROM	parent image to build from
ENV	sets environment variable <key> to value <value>
WORKDIR	sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile
COPY	copies new files/directories from <source> to filesystem of the container at the path <dest>
RUN	executes any commands in a new layer on top of the current image and commit the results.; the resulting committed image will be used for the next step in the Dockerfile.
CMD	provides defaults for an executing container; there can only be one CMD instruction in a Dockerfile
ENTRYPOINT	allows you to configure a container that will run as an executable; default is '/bin/bash -c'

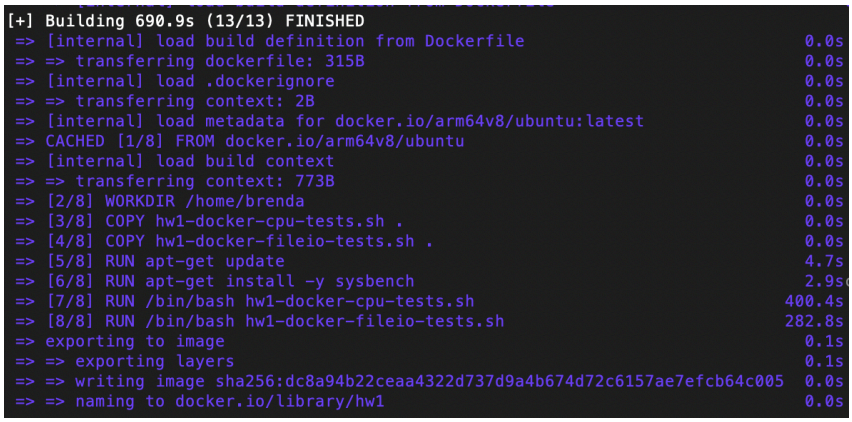
$ vim Dockerfile

```
# Dockerfile
# Create new image and run HW1 Benchmark Tests

FROM arm64v8/ubuntu

ENV user brenda

WORKDIR /home/$user

COPY hw1-docker-cpu-tests.sh .
COPY hw1-docker-fileio-tests.sh .

RUN apt-get update
RUN apt-get install -y sysbench

RUN /bin/bash hw1-docker-cpu-tests.sh
RUN /bin/bash hw1-docker-fileio-tests.sh
```

$ chmod u+x Dockerfile
$ docker build -t hw1 .
# -t: include tag in image name

$ docker images
# check that image was created locally

$ docker run -it --rm hw1 bash
# launch the container to check the results files of the benchmark scripts



Vagrantfile

https://github.com/ppggff/vagrant-qemu

$ brew install vagrant
$ vagrant --version
> Vagrant 2.3.4

$ vagrant plugin install vagrant-qemu

$ vagrant box add bytesguy/ubuntu-server-20.04-arm64
> choose ‘parallels’

$ find .vagrant.d/boxes -name '*bytesguy*' # find box dir

$ cd ~/.vagrant.d/boxes/bytesguy-VAGRANTSLASH-ubuntu-server-20.04-arm64/1.0.0/parallels/
# enter dir
$ find . -name ‘*.hds' # find the hard disk

$ qemu-img convert -c -f parallels -O qcow2 ./Ubuntu\ Server\ 20.04.pvm/harddisk1.hdd/harddisk1.hdd.0.{5fbaabe3-6958-40ff-92a7-860e329aab41\}.hds box.qcow2

convert to qcow2 image

$ cd -
$ vagrant init bytesguy/ubuntu-server-20.04-arm64 # places a ‘Vagrantfile’ in the directory
$ vim Vagrantfile
$ vagrant up --provider qemu # starts the virtual machine from Vagrantfile

```
(base) ➜  ubuntu_vm cat Vagrantfile
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|

  config.vm.box = "bytesguy/ubuntu-server-20.04-arm64"
  config.vm.box_version = "1.0.0"

  config.vm.synced_folder ".", "/vagrant_data"

  config.vm.provider "qemu" do |q|
        q.arch = "aarch64"
        q.cpu = "cortex-a57"
        q.machine = "virt,highmem=off"
        q.memory = "2048"
        q.image_path = "/Users/brendawang/.vagrant.d/boxes/bytesguy-VAGRANTSLASH-ubuntu-server-20.04-arm64/1.0.0/parallels/box.qcow2"
  end


  config.vm.provision "shell", inline: <<-SHELL

        apt-get update
        apt-get install -y sysbench
        bash '/vagrant_data/hw1-qemu-cpu-tests.sh'
        bash '/vagrant_data/hw1-qemu-fileio-tests.sh'

  SHELL
end
```

```
(base) ➜  ubuntu_vm vagrant up --provider qemu
Bringing machine 'default' up with 'qemu' provider...
==> default: Box 'bytesguy/ubuntu-server-20.04-arm64' could not be found. Attempting to find and install...
    default: Box Provider: libvirt
    default: Box Version: 1.0.0
==> default: Loading metadata for box 'bytesguy/ubuntu-server-20.04-arm64'
    default: URL: https://vagrantcloud.com/bytesguy/ubuntu-server-20.04-arm64
The box you're attempting to add doesn't support the provider
you requested. Please find an alternate box or use an alternate
provider. Double-check your requested provider to verify you didn't
simply misspell it.

If you're adding a box from HashiCorp's Vagrant Cloud, make sure the box is
released.

Name: bytesguy/ubuntu-server-20.04-arm64
Address: https://vagrantcloud.com/bytesguy/ubuntu-server-20.04-arm64
Requested provider: ["libvirt"]
```

* couldn’t load ubuntu installer in VirtualBox VM, apparently this just crashes on apple silicon machines