COEN 241 – Fall 2021
System vs. OS Virtualization

Abhay Tamilselvan (W1606871)

TABLE OF CONTENTS

Detailed configurations of experimental setup

The experiments for the VM machine and the docker will be conducted on a MacOS with Intel processor having 16GB RAM and a quad core processor.

The Ubuntu Virtual Machine will be mounted on a QEMU image with 10GB hard disk space. The number of CPUs used is 1 and the RAM size is 2 GB.

The docker container will also have memory limit of 2 GB RAM so that the the test environment is similar.

The tests has been conducted for both Ubuntu VM and container under 3 different scenarios -

CPU Tests

1. RAM = 2GB, threads = 1, cpu-max-prime = 20000
2. RAM = 2GB, threads = 1, cpu-max-prime = 5000000, max-time = 30s
3. RAM = 2GB, threads = 16, cpu-max-prime = 50000

File IO Tests

1. RAM = 2GB, threads = 16, file-total-size = 3GB, file-test-mode = RNDRW
2. RAM = 2GB, threads = 4, file-total-size = 2GB, file-test-mode = SEQRD
3. RAM = 2GB, threads = 8, file-total-size = 3GB, file-test-mode = SEQWR

Steps to enable a QEMU VM

1. As I am using a Mac with Intel chip, qemu can be installed using the command below.

```
brew install qemu
```

2. After installation, a QEMU image with a hard disk space of 10GB can be created using

```
sudo qemu-img create ubuntu.img 10G
```

3. After creating the image, we have to download the ubuntu ISO file (ubuntu-20.04.3-live-server-amd64) and then install the VM using the following command

```
sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom ./
ubuntu-20.04.3-live-server-amd64 -m 2G
```

QEMU arguments used in creating the VM

1. `-m` argument is used to assign the startup RAM size to the guest OS. I have used this to assign a RAM size of 1536MB.

2. `-boot` is used to boot the VM. a, b is for floppy disk, c is for hard disk and d is for cdrom. Hard disk boot is the default. We have used d here and considered the ubuntu ISO file as a cdrom.

3. `-hda` is used to use file as a hard disk image

4. `-cdrom` to use file as a cdrom image

Other useful arguments of QEMU have been tested on the terminal

1. `-smp` argument is used to simulate a SMP (shared memory processing) system with n CPUs. I created a guest OS with 4 CPUs using the below command.

```
sudo qemu-system-x86_64 -hda ubuntu.img -m 2G -smp 4
```

2. `-accel` argument is used to enable an accelerator. Different accelerators like kvm, xen, etc can be used. The default accelerator is tcg.

3. `-machine` can be used to select the emulated machine by name.

4. `-cpu` is used to choose among the different cpu models available.

Main steps to enable a Docker container

After installing docker, the following commands should be entered in the terminal to enable the docker image.

```
docker pull csminpp/ubuntu-sysbench
docker run -it —-memory="2g" csminpp/ubuntu-sysbench /bin/bash
```

`docker pull` is used to pull a docker image from the docker hub and store it on the local device.
`docker run` is used to to run the docker image.
`-it` flag attaches an interactive tty in the container.

Now, the bash terminal of the docker container is opened and we can run our experiments on it based on our requirements.

Other docker commands that are frequently used to manage docker images and containers are as follows:

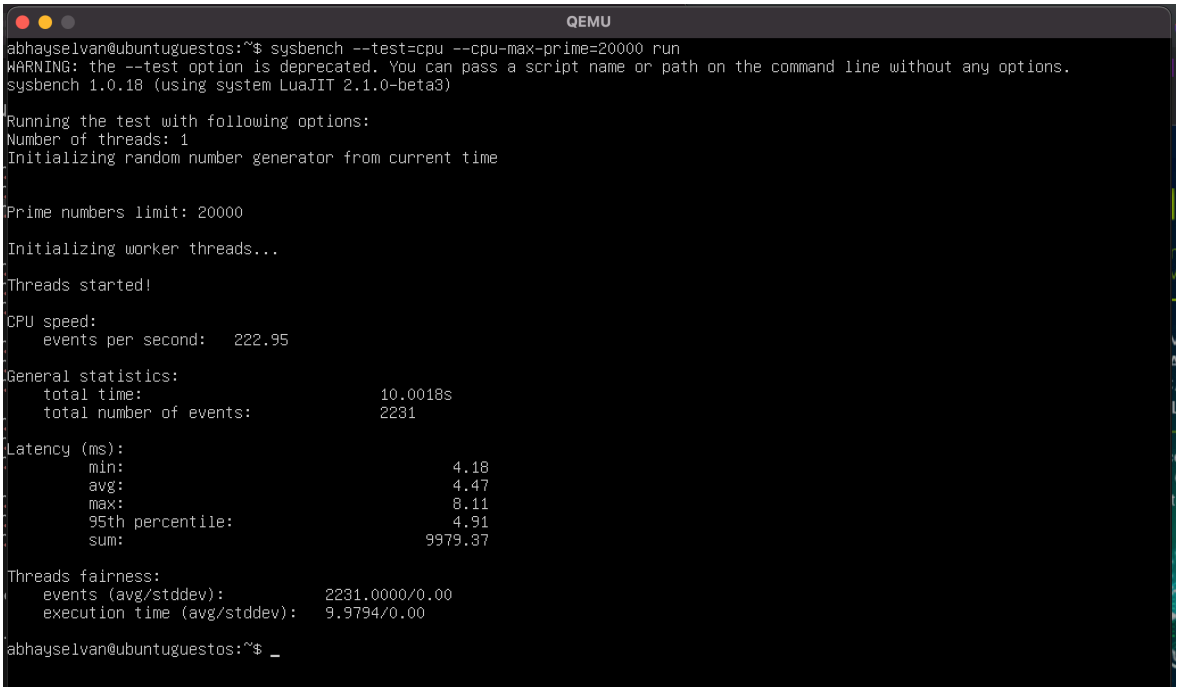
`docker images` - This command will display all the docker images on our local machine.
`docker ps` - This command will display all the docker containers that are currently running.
`docker rm` - This command is used to clean up the containers once we are done using it.
`docker container prune` - This command will remove all stopped containers.

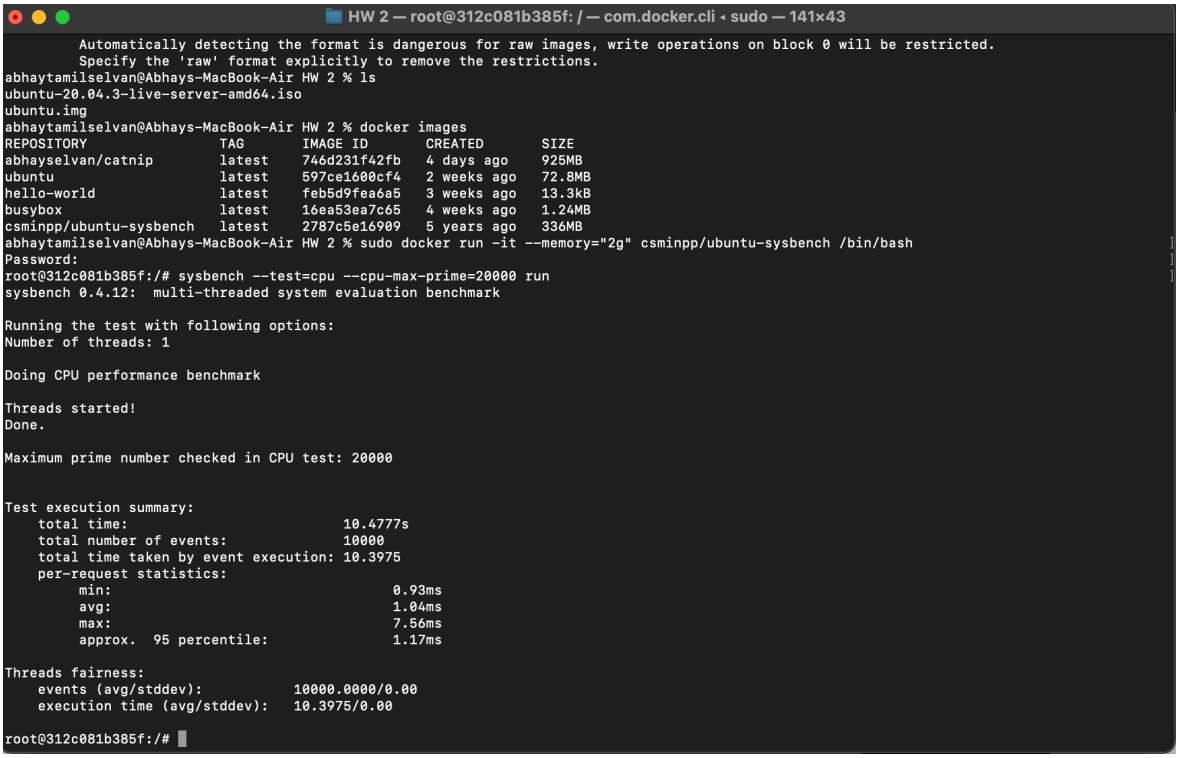
Proof of Experiment

I have conducted three experiments for CPU test and three experiments for File IO test on both the virtual machine and the container. For each experiment, I have run the test 5 times each. The screenshots of each experiment is posted below.

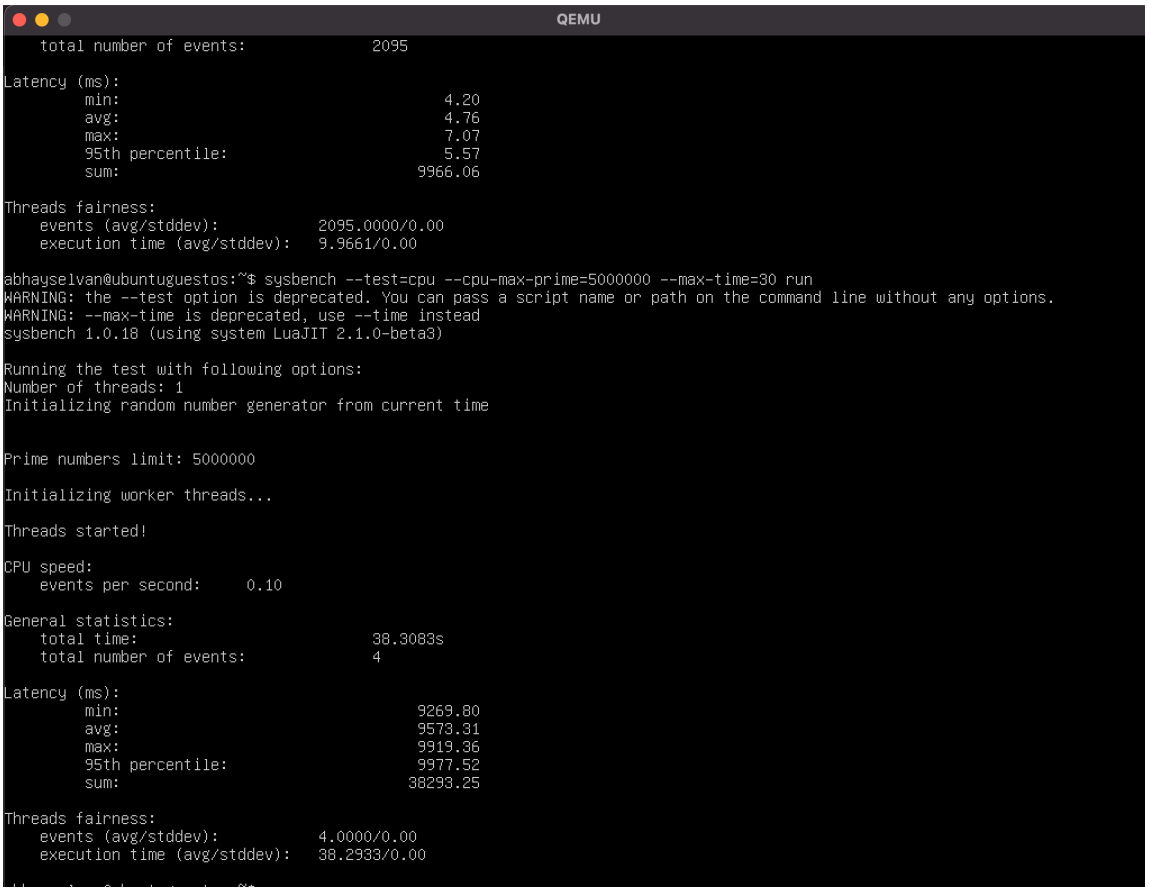
CPU TESTS

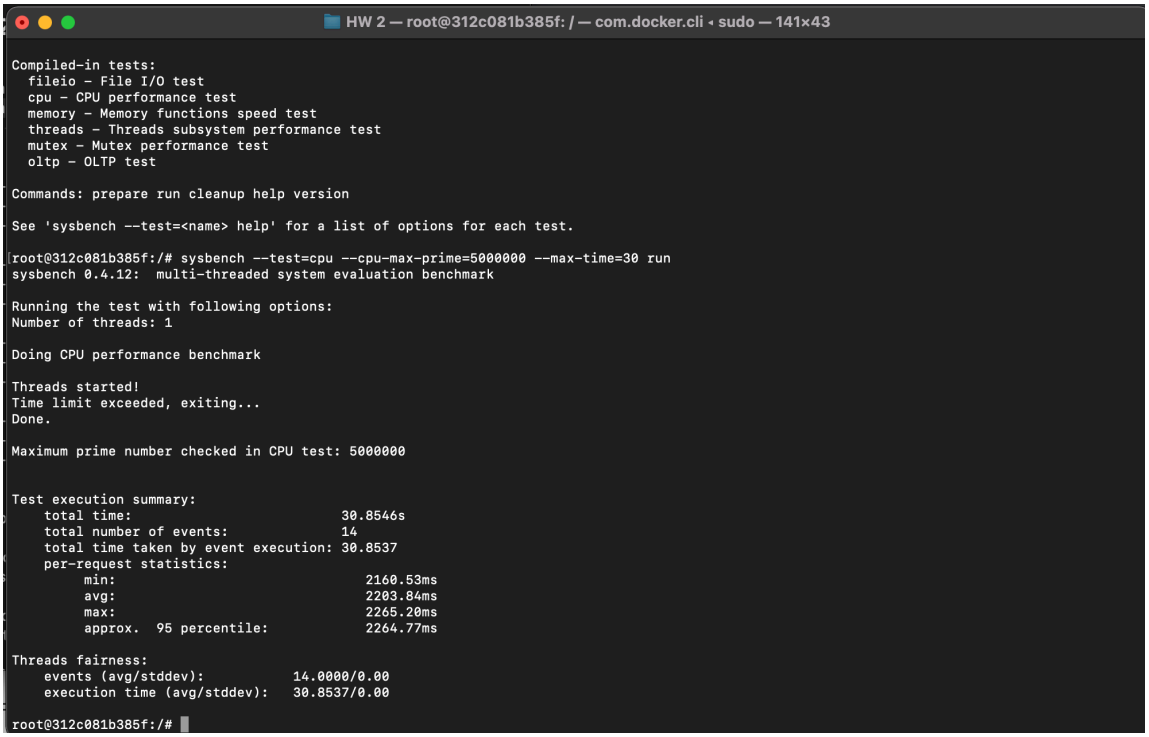
Test 1: `sysbench --test=cpu --cpu-max-prime=20000 run`



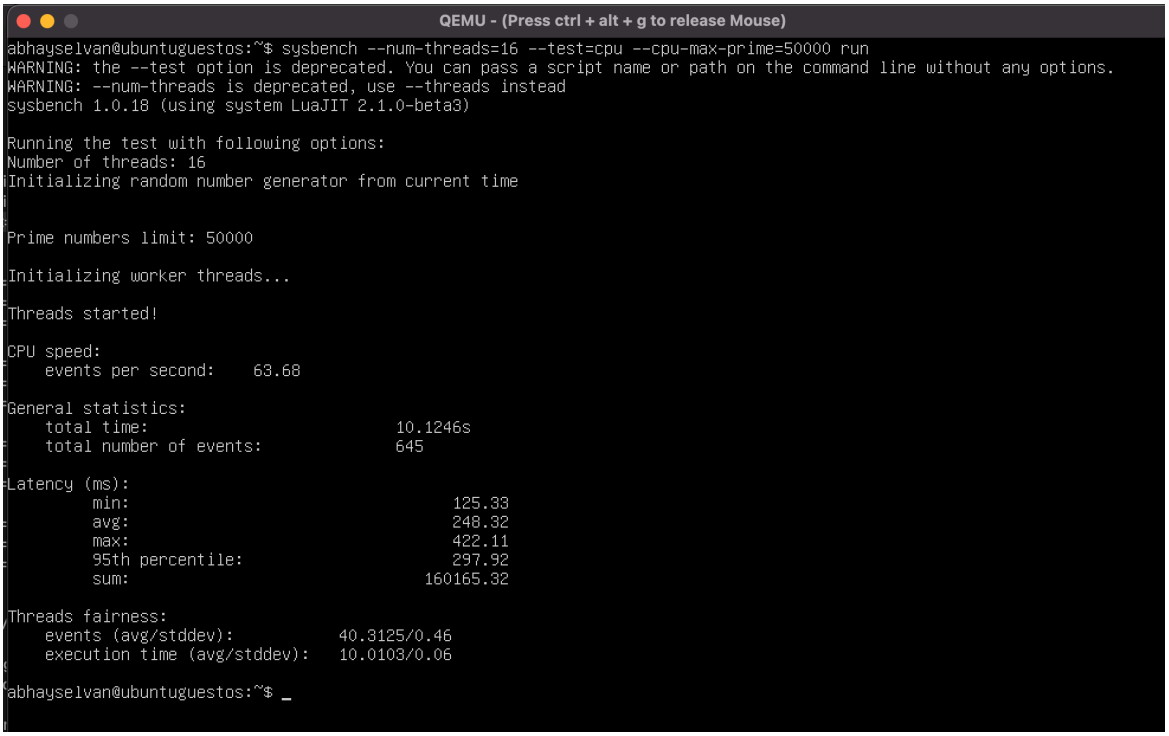


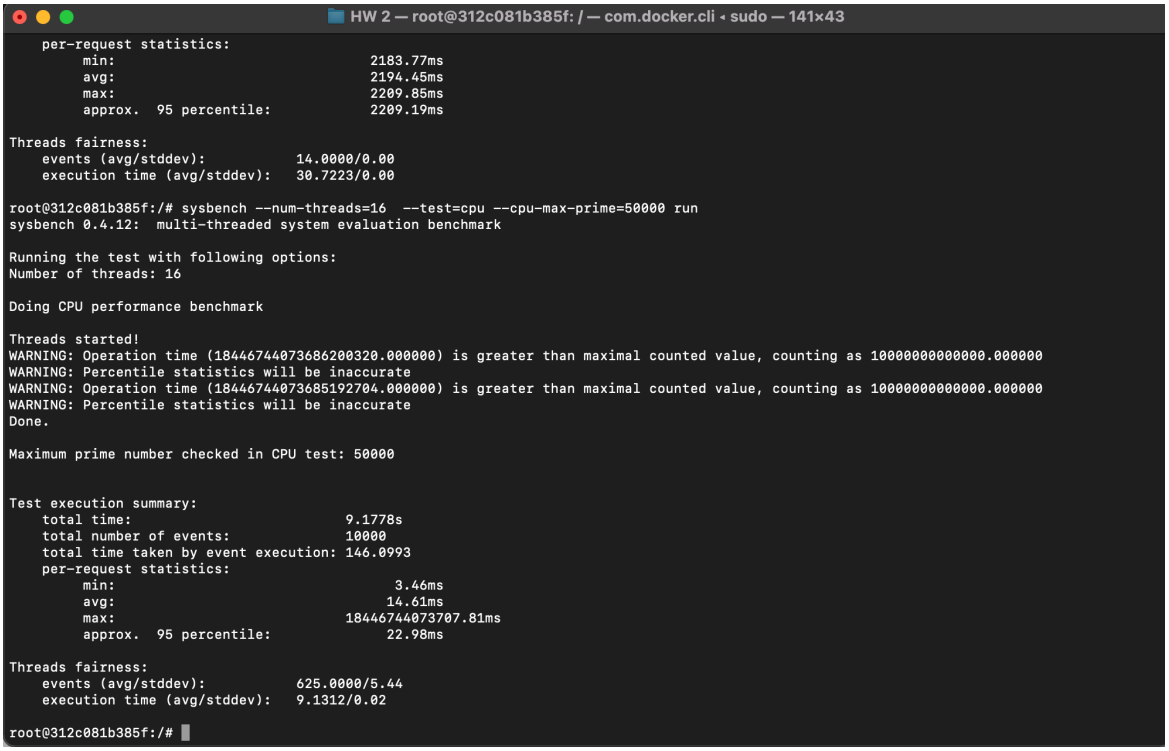
Test 2: `sysbench --test=cpu --cpu-max-prime=5000000 –max-time=30 run`





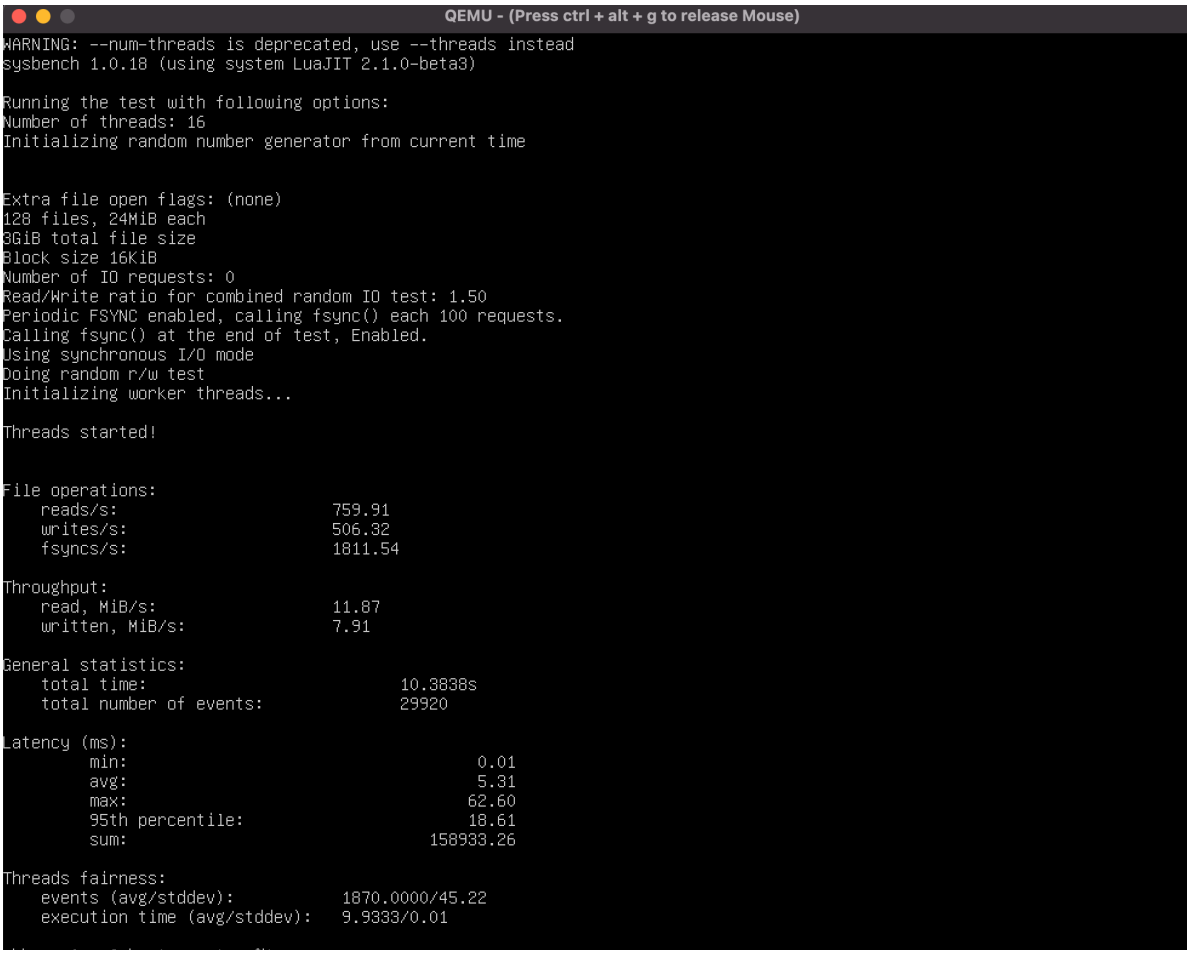
Test 3: `sysbench --num-threads=16  --test=cpu --cpu-max-prime=50000 run`

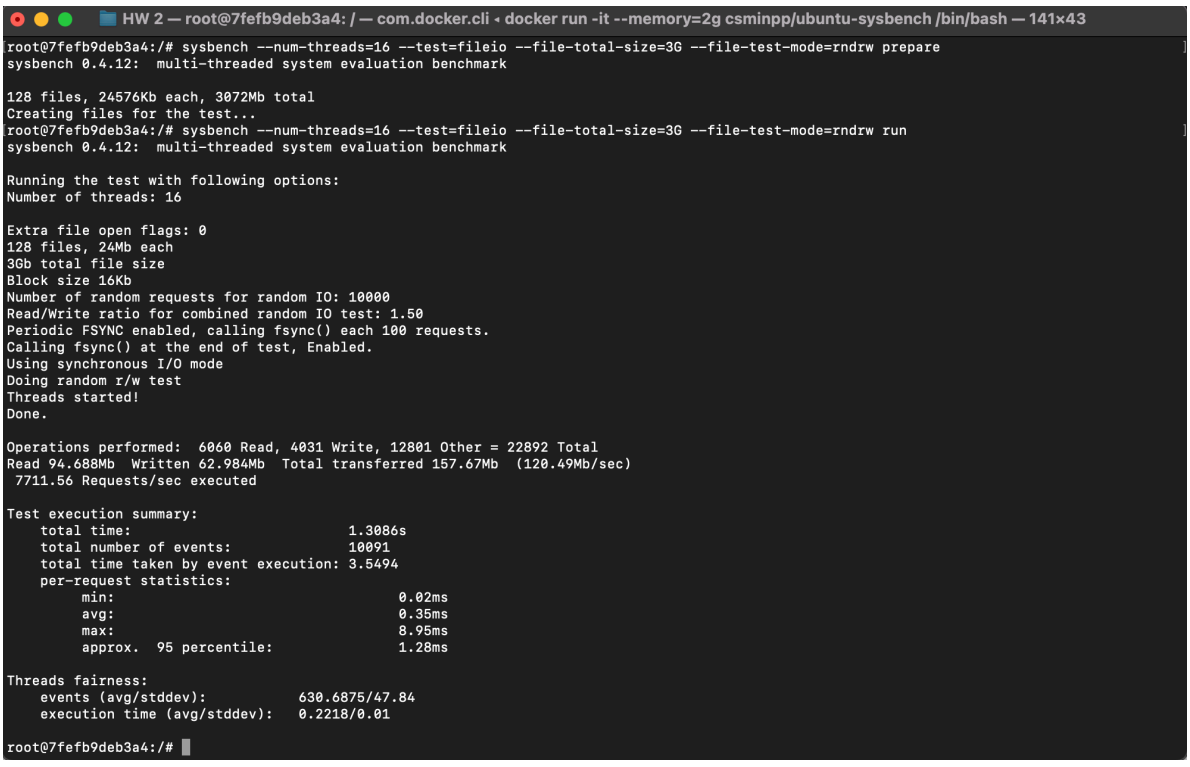




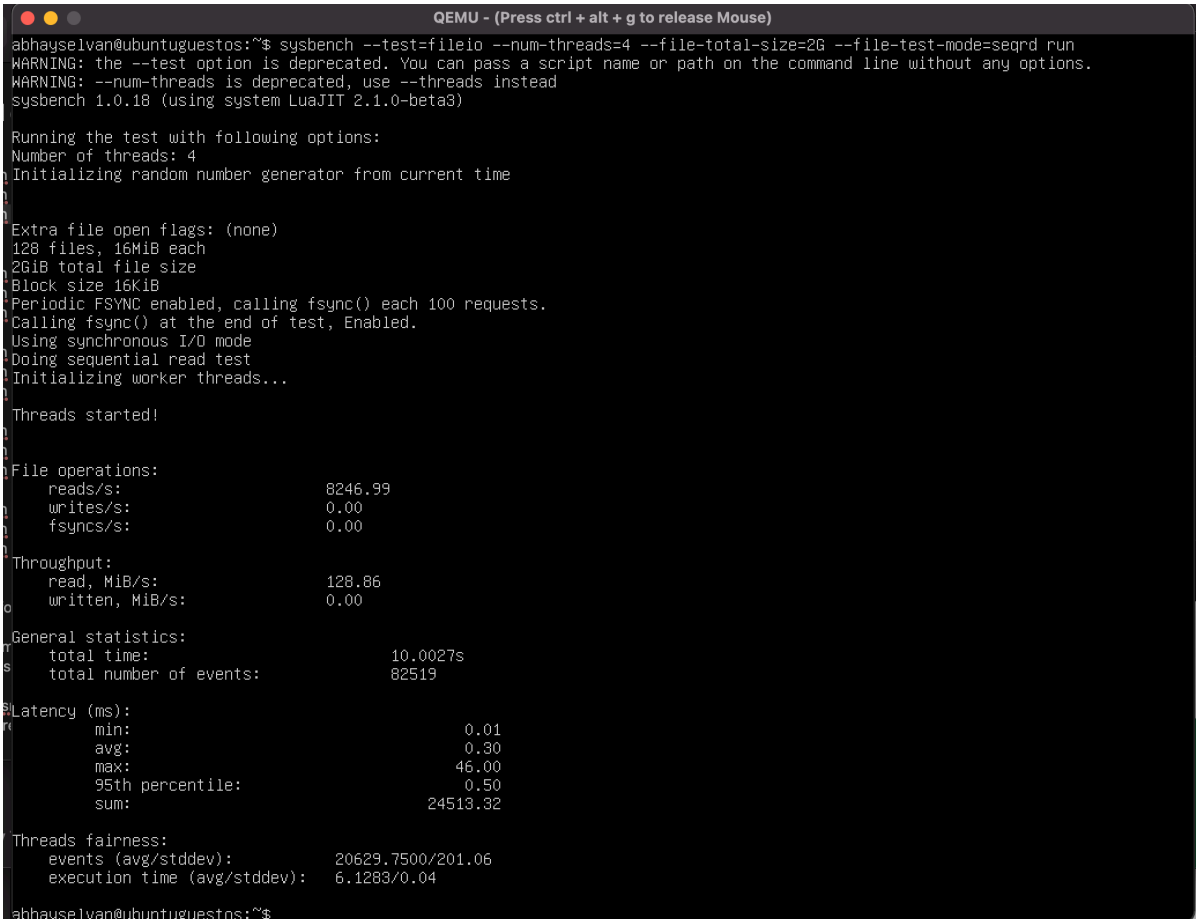
FILEIO TESTS

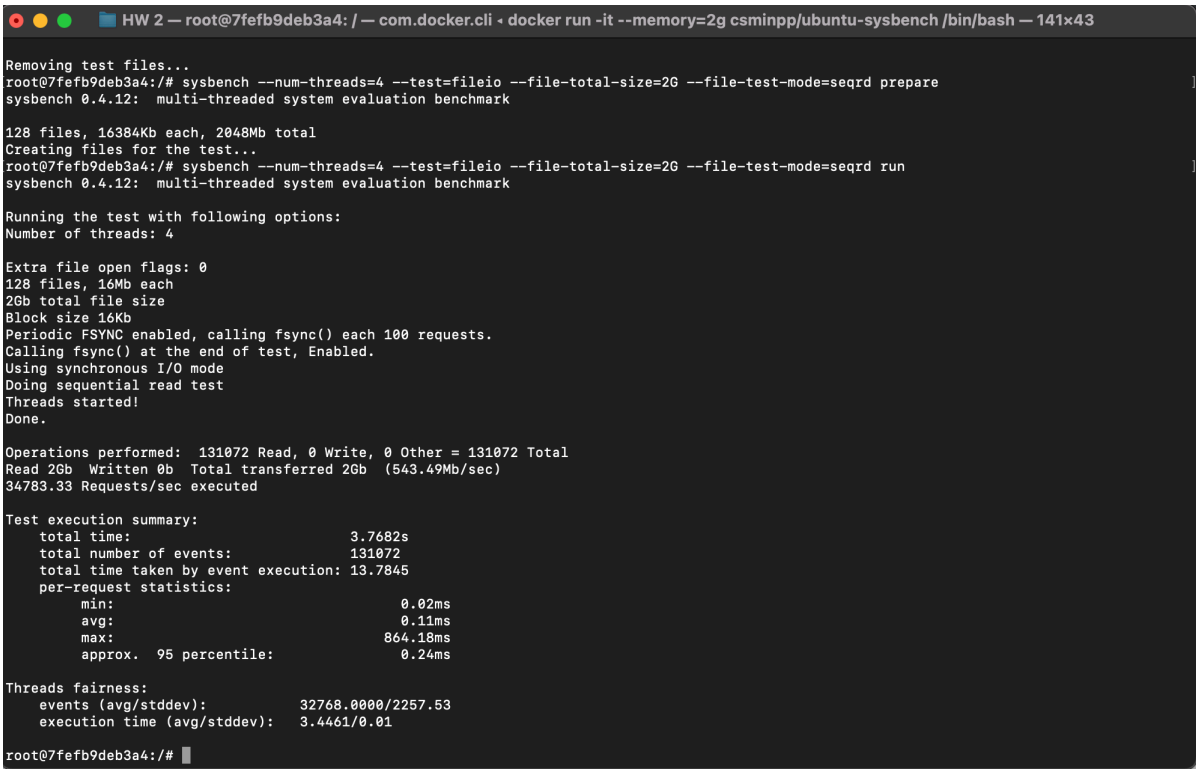
Test 1: sysbench --num-threads=16 --test=fileio --file-total-size=3G --file-test-mode=rndrw run



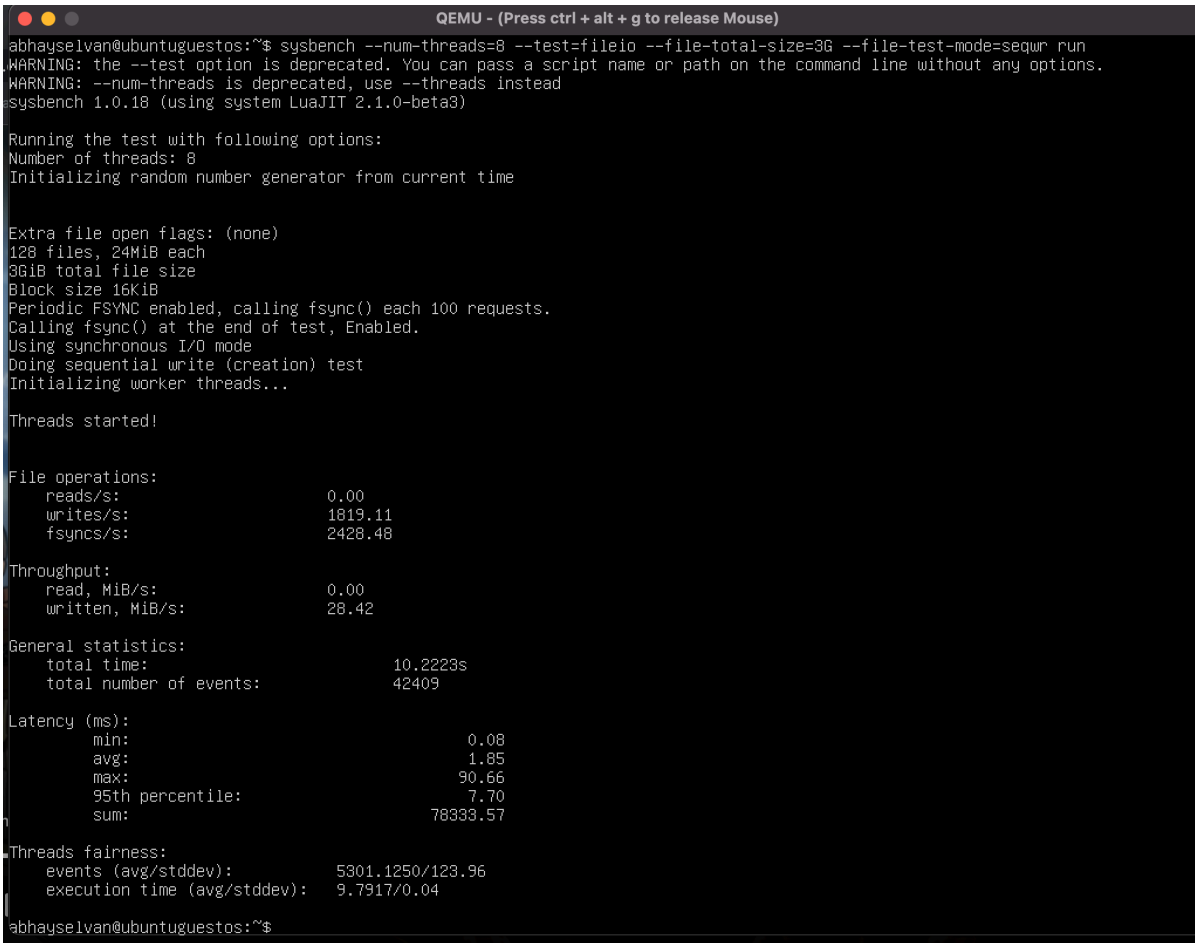


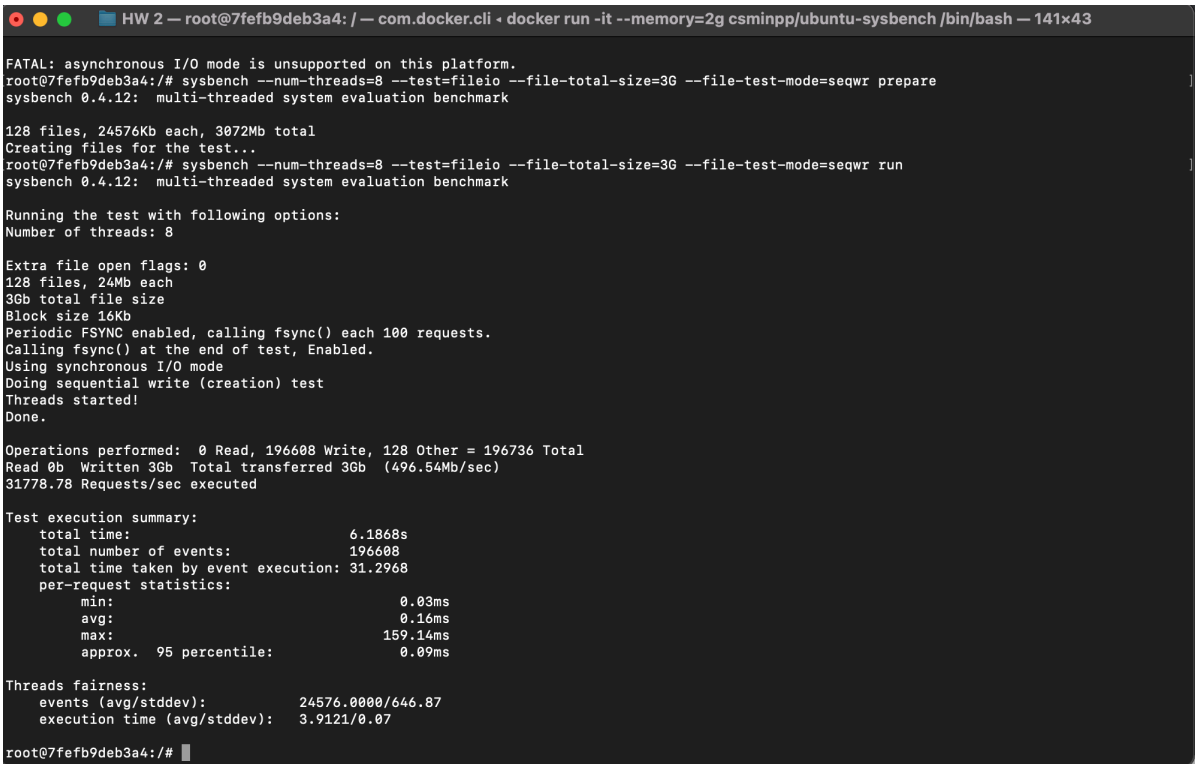
Test 2: sysbench --num-threads=4 --test=fileio --file-total-size=2G --file-test-mode=seqrd run





Test 3: sysbench --num-threads=8 --test=fileio --file-total-size=3G --file-test-mode=seqwr run





Shell Scripts

cputests.sh

```
#! /bin/bash
#

primes=("20000" "5000000" "50000")
times=("0" "30" "0")
threads=("1" "1" "16")
testruns=5
testcases=3

for ((i=0; i<$testcases;i++));
do
    for (( j=1; j <=$testruns; j++ ))
    do
        sysbench --num-threads=${threads[$i]} --test=cpu --cpu-max-
prime=${primes[$i]} --max-time=${times[$i]} run
    done
done
```

fileiotests.sh

```
#! /bin/bash
#

commands=("prepare" "run" "cleanup")
modes=("rndrw" "seqrd" "seqwr")
filesizes=("3G" "2G" "3G")
threads=("16" "4" "8")
testruns=5
testcases=3
teststages=3

for ((i=0; i<$testcases;i++));
do
    for (( j=0; j <$testruns; j++ ))
    do
        for (( k=0; k<$teststages; k++))
        do
            sysbench --num-threads=${threads[$i]} --test=fileio --
file-total-size=${filesizes[$i]} --file-test-mode=${modes[$i]} $
{commands[$k]}
        done
    done
done
```

Performance tools and analysis

Events per second has been chosen as unit of measurement for both CPU tests and File IO tests because two different versions of sysbench have been used for docker and QEMU VM respectively.

CPU Tests

For CPU tests, I have altered three values to generate comprehensive test results - CPU-max-prime value, max-time and num-threads.

`cpu-max-prime=20000`

	Average	Min	Max	Std
Docker	954.88	952.3	957.5	1.739
QEMU VM	215.71	209.26	222.95	5.256

`cpu-max-prime=5000000 —max-time=30`

	Average	Min	Max	Std
Docker	0.455	0.453	0.456	0.009
QEMU VM	0.104	0.102	0.105	0.013

`cpu-max-prime=50000 num-threads=16`

	Average	Min	Max	Std
Docker	1039.26	947.66	1085.39	55.68
QEMU VM	63.27	61.75	64.03	0.815

File IO Tests

While conducting File IO tests, it is important to use the flag —file-extra-flags=direct to avoid caching of files in the memory. We should also allocate —file-total-size greater than that of the total memory just to make sure that caching doesn't happen despite our best efforts.

`num-threads=16--file-total-size=3G --file-test-mode=rndrw`

	Average	Min	Max	Std
Docker	8057.95	7711.56	8270.26	192.12
QEMU VM	2878.5	2756.63	2946.41	74.29

`num-threads=4--file-total-size=2G --file-test-mode=seqrd`

	Average	Min	Max	Std
Docker	40517.66	34783.33	45787.02	3767.68
QEMU VM	10011.33	9208.67	10526.14	464.69

`num-threads=8--file-total-size=3G —file-test-mode=seqwr`

	Average	Min	Max	Std
Docker	32044.89	31556.37	32474.51	462.03
QEMU VM	4019.25	3686.14	4212.16	203.13

Inference:

- Docker container is faster than QEMU VM in all three CPU tests
- Docker container is faster than QEMU VM in all three File IO tests
- Events per second decreases when max prime value increases for both docker container and QEMU VM.
- Increasing the number of threads results in increase in performance of the docker container for CPU tests but the QEMU VM's performance doesn't increase correspondingly.
- The sequential file IO operations have better performance compared to random file IO operations

Analysis:

Containers faster than VMs

Containers have better performance than Virtual Machines which is expected because they only provide OS virtualization and don't interact directly with the hardware. Even installation of containers is much faster than Virtual Machines making it is easier to deploy and share across platforms. This is because a guest OS needs to be installed for a VM whereas the host OS itself is used by containers.

Increasing number of threads increases performance

When we increase the number of threads, more instructions are processed by the CPU concurrently. But this doesn't mean that randomly increasing threads would result in a better performance. It depends on the number of cores that a machine has and the instructions themselves being non-sequential.

Sequential I/O faster than random I/O

The sequential I/O is faster than random I/O because of the way disk drives work. To access a file, the disk head needs to seek the location of the file. If a file is stored sequentially, then time is saved on the seek operation while the read/write operation take the same time for both random and sequential operations.

Events per second decreases with complexity of task to be performed

An event is the task that the system needs to perform. So, if it's a very simple task, then it is completed quickly. So more numbers of events can be completed per second if it's an easy task. Thus, it can be easily inferred that events per second decreases with increase in the complexity of a task.

CPU Utilization

CPU usage has been measured on the docker container as well as the host OS during idle time and sysbench cpu benchmark.

Container

We need to create a new terminal for the same docker container so that we can test the CPU usage while the sysbench tests are running. We can do that by getting the container id of the running container and using the following command:

```
docker exec –it <container id> /bin/bash
```

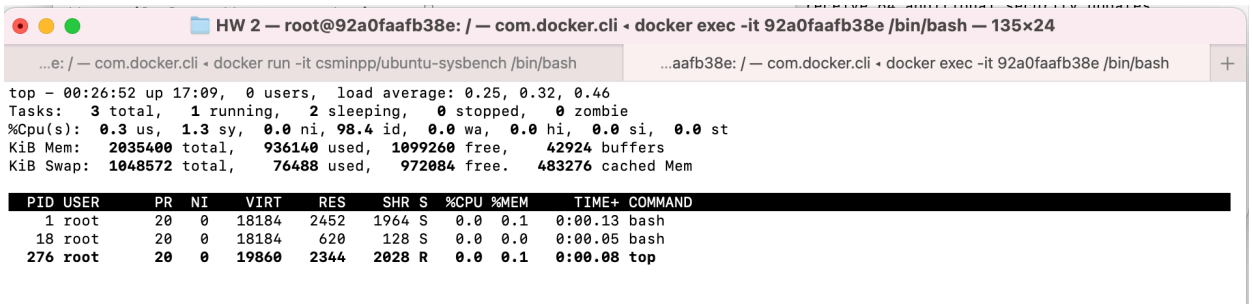
`top` command has been used to check the CPU usage on the docker container.

In %cpu(s),

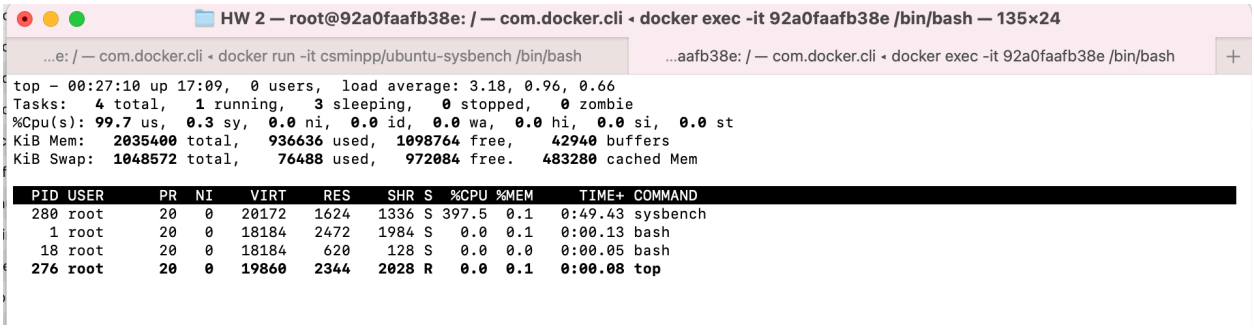
- us - stands for user level CPU usage
- sy - stands for kernel level CPU usage
- id - stands for idle CPU usage (% of available CPU that is idle)

The PIDs of the processes and their usage are also displayed below.

When docker is idle



When docker is running sysbench cpu benchmark test



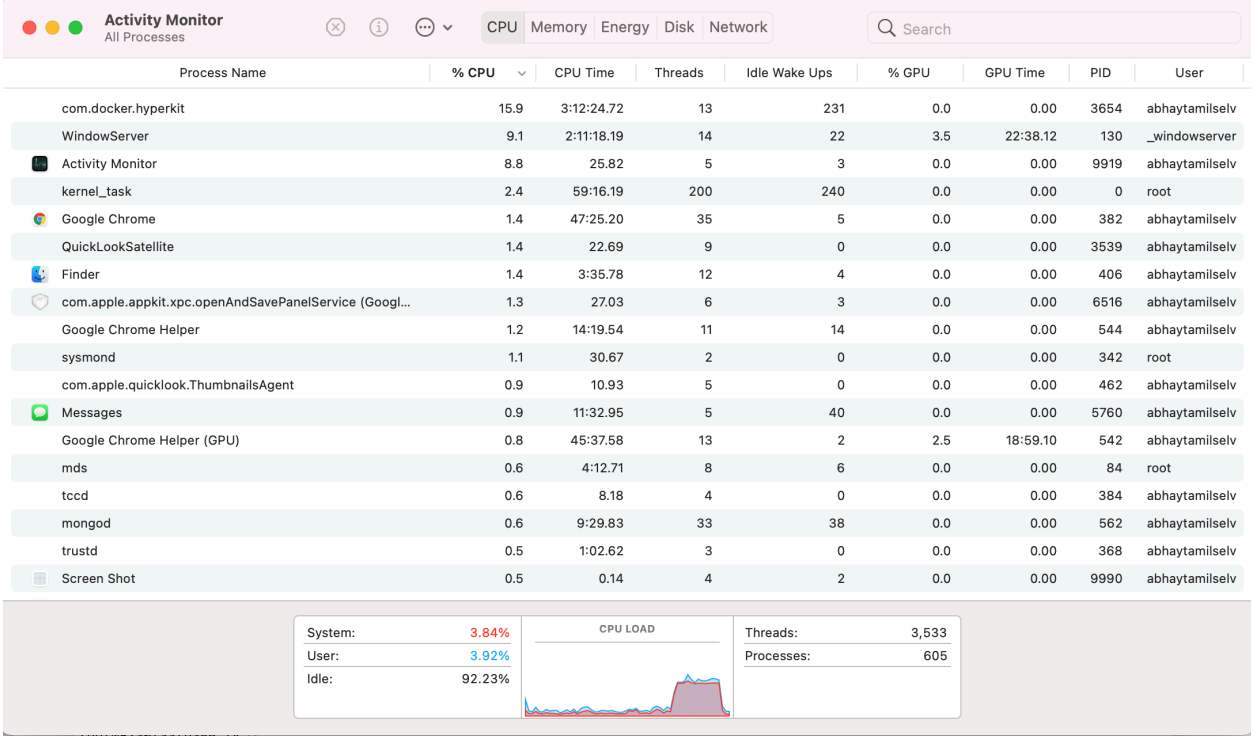
	Sysbench not running	Sysbench CPU test running
User level CPU usage %	0.3	99.7
Kernel level CPU usage %	1.3	0.3
Idle CPU %	98.4	0.0

As can be seen from the table above, when sys bench CPU benchmark test is being run, the user level CPU usage is close to 100%. The kernel level is barely used.

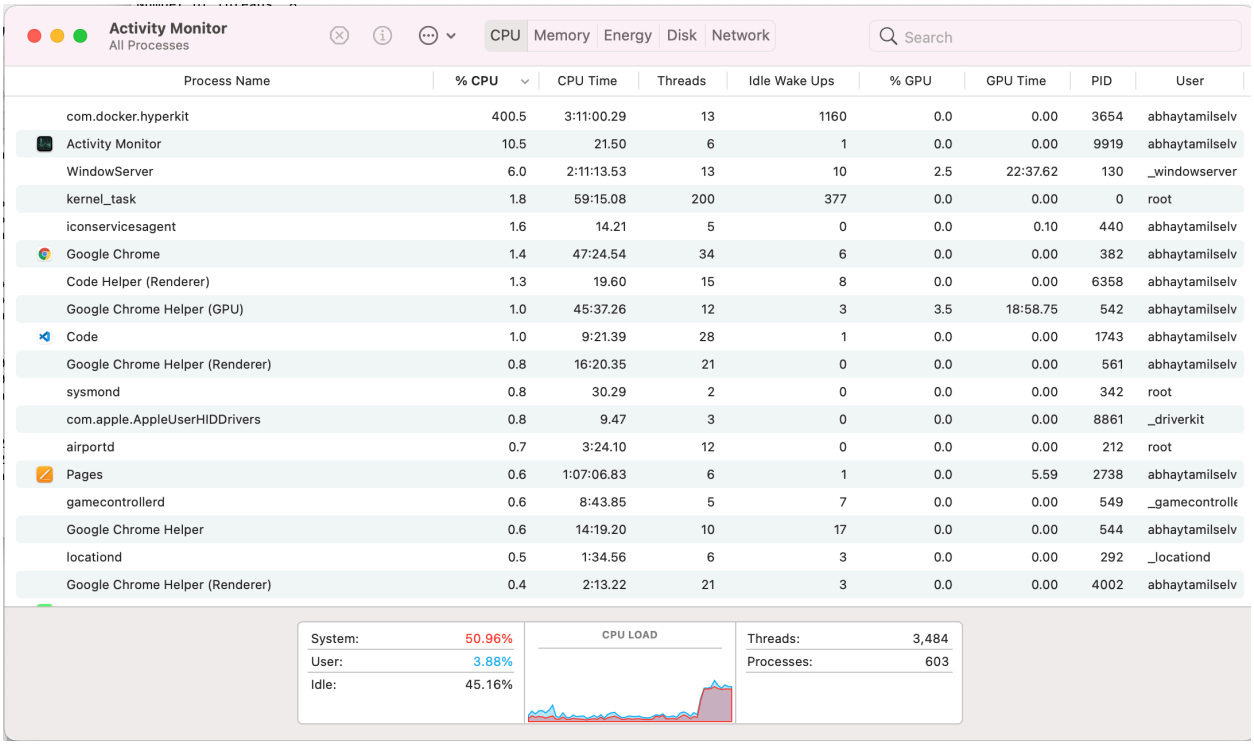
`Host OS`

When we look at the CPU usage in the host OS, the following CPU usage was observed.

When sysbench is not running on docker



When sysbench is running CPU benchmark test on docker



	Sysbench not running	Sysbench CPU test running
User level CPU usage %	3.92	3.88
Kernel level CPU usage %	3.84	50.96
Idle CPU %	92.23	45.16

In the host OS, the kernel level CPU usage jumps from 3.84% to 50.96% when we run sysbench CPU benchmark test. The docker container when idle has a CPU usage of 15.9% which jumps to 400% when sysbench is run. One difference to note here is that inside the docker, user level CPU usage is accessed for the process whereas in the host OS, the kernel level CPU usage is used. ***This shows that the container uses the kernel of the host OS***.

I/O Utilization

I/O utilization of a VM or container can be measured using the `iostat` command. To make use of this command, we need to install `sysstat` which can be done using the command:

```
sudo apt install sysstat
```

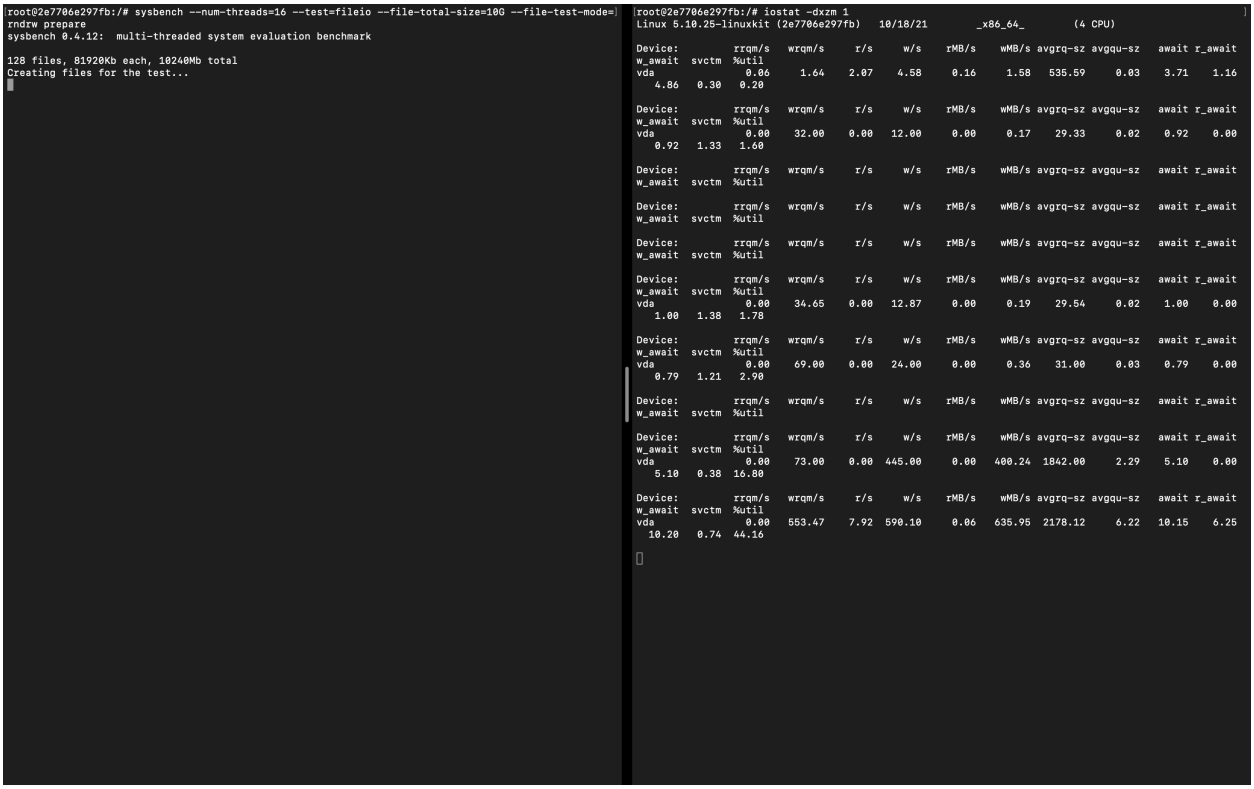
After installing sysstat, we need to use the command,

```
iostat -dxzm 1
```

The -d flag is used to show disk utilization only. -x is used to show extended columns, -z is used to skip devices with zero metrics. The 1 here denotes that the disk utilization report would be displayed on the command line every 1 second. After running the above commands, we can run the sysbench fileIO test on a different terminal of the same container to see how disk utilization changes when we run the test.

The screenshots for the disk utilization during the three phases of the sysbench fileio test are as follows

During prepare stage:



During run stage:

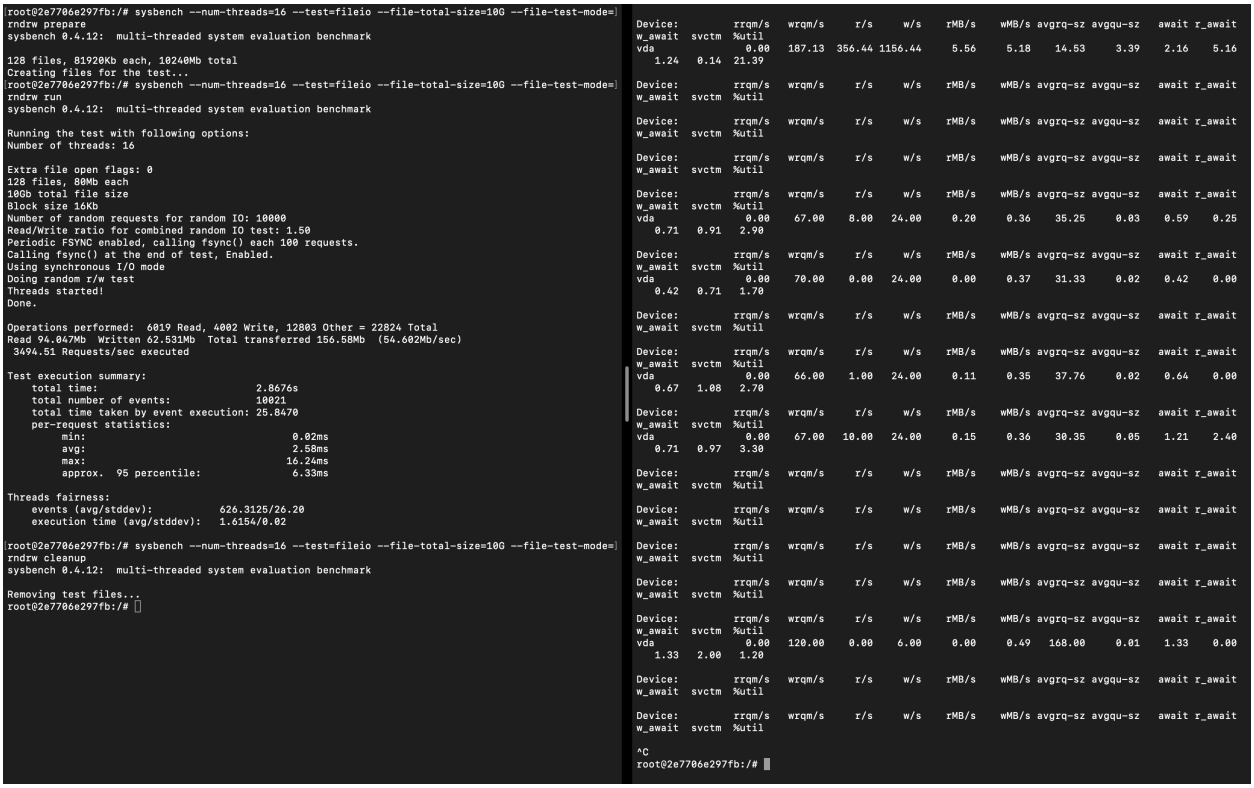
```
root@2e7706e297fb:/# sysbench --num-threads=16 --test=fileio --file-total-size=10G --file-test-mode=
rndrw prepare
sysbench 0.4.12:  multi-threaded system evaluation benchmark

128 files, 81920Kb each, 10240Mb total
Creating files for the test...
root@2e7706e297fb:/# sysbench --num-threads=16 --test=fileio --file-total-size=10G --file-test-mode=
rndrw run
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 16

Extra file open flags: 0
128 files, 80Mb each
10Gb total file size
Block size 16Kb
Number of random requests for random IO: 10000
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Threads started!
```

```
   6.15   0.51  49.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   138.00    0.00  880.00     0.00   889.04  2069.05     5.27    5.84    0.00
   5.84   0.51  44.70

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   102.00    0.00  815.00     0.00   753.62  1893.77     8.17    9.82    0.00
   9.82   0.64  51.90

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   142.00    0.00 1073.00     0.00  1004.80  1917.82     6.28    5.81    0.00
   5.81   0.38  40.60

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   125.00    2.00  961.00     0.01   880.50  1872.56     6.62    6.78    3.50
   6.79   0.48  46.10

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   149.00    0.00 1069.00     0.00  1003.04  1921.64     6.32    5.87    0.00
   5.87   0.38  40.50

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   134.00    0.00 1023.00     0.00   950.04  1901.94     5.87    5.65    0.00
   5.65   0.41  42.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00    73.00    0.00  519.00     0.00   528.31  2084.75     4.90    9.38    0.00
   9.38   0.51  26.70

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   568.00 1307.00 3685.00    24.77    16.28    16.84     9.36    1.80    4.25
   0.94   0.13  67.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               1.00  1290.00 1985.00 6053.00    31.08    28.31    15.13    14.79    1.77    4.23
   0.96   0.12  99.90
```

```
root@2e7706e297fb:/# sysbench --num-threads=16 --test=fileio --file-total-size=10G --file-test-mode=
rndrw prepare
sysbench 0.4.12:  multi-threaded system evaluation benchmark

128 files, 81920Kb each, 10240Mb total
Creating files for the test...
root@2e7706e297fb:/# sysbench --num-threads=16 --test=fileio --file-total-size=10G --file-test-mode=
rndrw run
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 16

Extra file open flags: 0
128 files, 80Mb each
10Gb total file size
Block size 16Kb
Number of random requests for random IO: 10000
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Threads started!
Done.

Operations performed:  6019 Read, 4002 Write, 12803 Other = 22824 Total
Read 94.047Mb  Written 62.531Mb  Total transferred 156.58Mb  (54.602Mb/sec)
 3494.51 Requests/sec executed

Test execution summary:
    total time:                          2.8676s
    total number of events:              10021
    total time taken by event execution: 25.8470
    per-request statistics:
         min:                                  0.02ms
         avg:                                  2.58ms
         max:                                 16.24ms
         approx.  95 percentile:               6.33ms

Threads fairness:
    events (avg/stddev):           626.3125/26.20
    execution time (avg/stddev):   1.6154/0.02

root@2e7706e297fb:/#
```

```
Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   125.00    2.00  961.00     0.01   880.50  1872.56     6.62    6.78    3.50
   6.79   0.48  46.10

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   149.00    0.00 1069.00     0.00  1003.04  1921.64     6.32    5.87    0.00
   5.87   0.38  40.50

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   134.00    0.00 1023.00     0.00   950.04  1901.94     5.87    5.65    0.00
   5.65   0.41  42.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00    73.00    0.00  519.00     0.00   528.31  2084.75     4.90    9.38    0.00
   9.38   0.51  26.70

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   568.00 1307.00 3685.00    24.77    16.28    16.84     9.36    1.80    4.25
   0.94   0.13  67.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               1.00  1290.00 1985.00 6053.00    31.08    28.31    15.13    14.79    1.77    4.23
   0.96   0.12  99.90

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00  1267.00 1906.00 5914.00    29.78    27.58    15.02    15.68    1.93    4.41
   1.13   0.13  99.90

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
vda               0.00   187.13  356.44 1156.44     5.56     5.18    14.53
   1.24   0.14  21.39

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avg
w_await  svctm  %util

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await
w_await  svctm  %util
```

During cleanup:



I/O latency is the time taken from request to completion of I/O request. It is measured using the await value in the iostat report. Await displays the average time (in milliseconds) for I/O requests issued to the device to be served.

I/O throughput is the data transfer speed in megabytes per second. It is measured using the wMBs value in the iostat report. wMBs is the number of megabytes written to the device per second.

I/O disk utilization is the percentage of CPU time during which I/O requests were issued to the device. We can measure this using the %util value.

	Prepare stage	**Run stage**	**Idle/cleanup stage**
I/O Latency	~6 milliseconds	~6 milliseconds	N/A
I/O Throughput	~900 MBPS	~900MBPS	N/A
I/O Disk Utilization	~50%	~100%	0 - 1%

From the table, the I/O latency for the device is around 6 milliseconds. The I/O throughput is around 900 MBPS. The disk utilization during prepare stage is around half of the total bandwidth available. During run stage, it uses close to 100% of the bandwidth (device saturation occurs this value is close to 100%). During cleanup or idle stage, there is close to zero disk utilization.

Vagrant File for Virtual Box

```
Vagrant.configure("2") do |config|
  config.vm.box = "bento/ubuntu-20.04"

  # Provider Settings
  config.vm.provider "virtualbox" do |vb|
    vb.memory = 2048
    vb.cpus = 4
  end
  # Network Settings
  # config.vm.network "forwarded_port", guest: 80, host: 8080
  # config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip:
"127.0.0.1"
  # config.vm.network "private_network", ip: "192.168.33.10"
  # config.vm.network "public_network"

  # Folder Settings
  config.vm.synced_folder ".", "/vagrant_data"

  # Provision Settings
  config.vm.provision "shell", path: "vagrantbootstrap.sh"

end
```

vagrantbootstrap.sh

```
#! /bin/bash
#

CPUTEST_PATH="/vagrant_data/cputests.sh"
FILEIOTEST_PATH="/vagrant_data/fileiotests.sh"

sudo apt-get update
sudo apt-get install -y sysbench
chmod +x /vagrant_data/cputests.sh
chmod +x /vagrant_data/fileiotests.sh

"$CPUTEST_PATH"
"$FILEIOTEST_PATH"
```

Dockerfile

```
FROM csminpp/ubuntu-sysbench

COPY dockerbootstrap.sh /dockerbootstrap.sh
COPY cputests.sh /cputests.sh
COPY fileiotests.sh /fileiotests.sh

RUN chmod +x dockerbootstrap.sh
RUN chmod +x cputests.sh
RUN chmod +x fileiotests.sh

ENTRYPOINT bash dockerbootstrap.sh
```

dockerbootstrap.sh

```
#! /bin/bash
#

CPUTEST_PATH="/cputests.sh"
FILEIOTEST_PATH="/fileiotests.sh"

"$CPUTEST_PATH"
"$FILEIOTEST_PATH"
```