COEN 241: Homework 1

Name: Juhi Checker
SCU ID: 1605378

Detailed configurations of your experimental setup

My System Setup:

System: Macbook Pro(15 inch, 2016)
Processor: 2.7 GHz Quad-core Intel Core i7
Memory: 16Gb 2133 MHz LPDDR3

QEMU experimental setup

I have decided to go with 3 experimental setup for QEMU. I have made used of two flags to setup the below setup. First is "-m" for memory and "-smp" for number of cores:

1. 3GB memory and 1 core
2. 6GB memory and 2 cores
3. 9GB memory and 4 cores

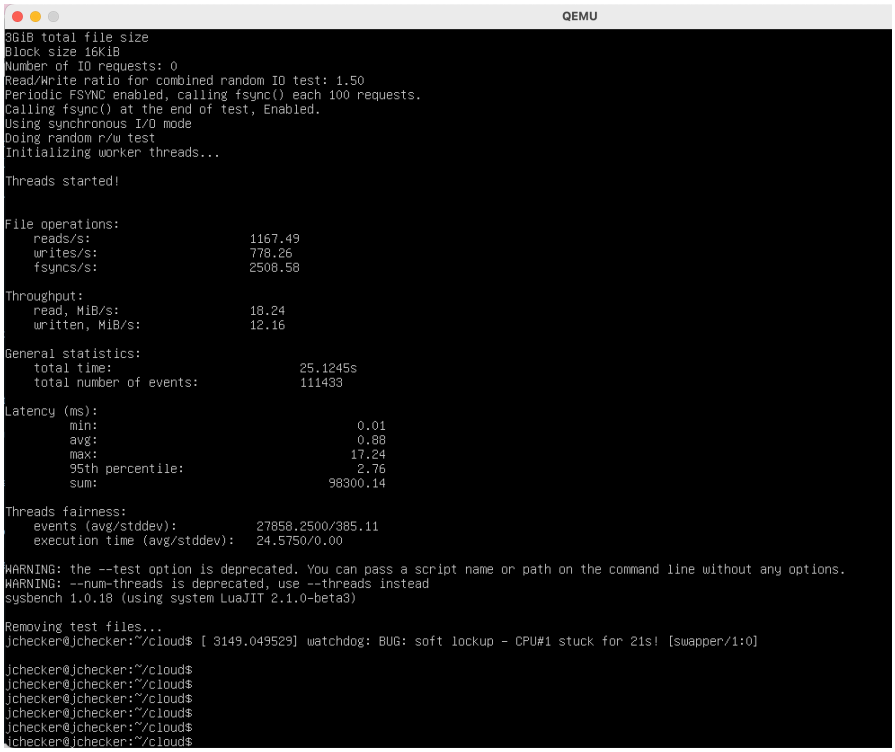
Docker experimental setup

In order to keep the testing configuration same as QEMU I have 3 experimental setup for Docker as well. I have used the 2 flags to set the memory and number of cores being used. These are "--memory" and "--cpuset-cpus" respectively.

1. 3GB memory and 1 core
2. 6GB memory and 2 cores
3. 9GB memory and 4 cores

Steps taken while installing QEMU

1. Downloading the ISO Ubuntu image 20.04 server from the link given in the assignment.
2. Installing QEMU by using this command
    a. brew install qemu
3. Creating the QEMU image by using this command. Since I had less space on my disk I gave only 10 GB space and kept my image name as ubuntu.img
    a. qemu-img create -f raw -o size=10G ubuntu.img
4. Installing ISO file into the ubuntu.img
    a. qemu-system-x86_64 -boot d -cdrom /Users/juhichecker/Downloads/ubuntu-20.04.5-live-server-amd64.iso -m 2048M -hda ubuntu.img
    b. You would be asked to enter username and password for the system so please enter them and I went with all the default settings to setup the system.
5. To run QEMU image and start run the following command
    a. sudo qemu-system-x86_64 -hda ubuntu.img -boot c -m 6G -smp 2 -boot strict=on
    b. Enter laptop password
    c. Wait for the setup of QEMU window. After entering your username and password you will see a screen below.



Steps taken while installing Docker

1. In order to install Docker, I used the link provided in the assignment for my computer.
2. After installing Docker, I used this command to check if there were any images existing in the docker.
    a. docker images
3. Pulling ubuntu image. I downloaded Ubuntu:focal image since it supported apt-get commands and allowed me to download sysbench. I did the following using the below command.
    a. docker pull ubuntu:focal

```
(base) Juhis-MacBook-Pro:~ juhichecker$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hw1                 latest              f6a19e22fafc        47 hours ago        134MB
trial               latest              890524d89d37        47 hours ago        72.8MB
ubuntu              kinetic-20220922    2793294e6a90        11 days ago         70.2MB
amd64/ubuntu        latest              216c552ea5ba        11 days ago         77.8MB
ubuntu              latest              216c552ea5ba        11 days ago         77.8MB
ubuntu              focal               817578334b4d        11 days ago         72.8MB
(base) Juhis-MacBook-Pro:~ juhichecker$
```

4. Creating your own image using ubuntu:focal as base image.
    a. The following command runs the image in the background and keeps it running using the “-d” flag
        i. docker run -d ubuntu:focal
    b. To check if your image was pulled run the following command
        i. docker images

```
      brave_chaum
(base) Juhis-MacBook-Pro:~ juhichecker$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hw1                 latest              f6a19e22fafc        2 days ago          134MB
trial               latest              890524d89d37        2 days ago          72.8MB
ubuntu              kinetic-20220922    2793294e6a90        11 days ago         70.2MB
amd64/ubuntu        latest              216c552ea5ba        11 days ago         77.8MB
ubuntu              latest              216c552ea5ba        11 days ago         77.8MB
ubuntu              focal               817578334b4d        11 days ago         72.8MB
(base) Juhis-MacBook-Pro:~ juhichecker$
```

c. Run the following command to check which images are currently running. You should see the ubuntu:focal
    i. docker ps

```
...emu-system-x86 ◂ sudo | ~ — -bash | ~ — -bash | ...— docker run -it hw1
(base) Juhis-MacBook-Pro:~ juhichecker$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
      NAMES
1b9a80acabac        hw1                 "bash"              16 hours ago        Up 16 hours
      brave_chaum
(base) Juhis-MacBook-Pro:~ juhichecker$
```

d. Enter the bash shell to install sysbench by running this command with the eeb4a being the container id. Output of the below command will take you to the root of the container
    i. docker exec -it eeb4a
e. Install sysbench in it by following the steps bellow
    i. apt-get update
    ii. apt-get install sudo
        1. This step because sudo is not already installed in the image
    iii. sudo apt-get install sysbench
    iv. sysbench –version

```
juhichecker — root@1b9a80acabac: /cloud — docker run -it hw1 — 100×47
...emu-system-x86 ◂ sudo | ~ — -bash | ~ — -bash | ...— docker run -it hw1
root@1b9a80acabac:/cloud# sysbench --version
sysbench 1.0.18
root@1b9a80acabac:/cloud#
```

5. To create a new image with the following updates I followed the below commands
    a. To stop the container
        i. docker container stop eeb4
    b. Committing all the changes to a new image, named “hw1”
        i. docker commit eeb4 hw1
    c. To check if the image was created
        i. docker images
    d. To run the new image. You might also run the command “sysbench --version” to check if the changes are saved in the new image.
        i. docker run -it hw1

<u>Experiments</u>

Experiments are conducted in the following manner for all the experimental setup:

1. CPU mode (using two flags “—cpu-max-prime” and “--time”)
    a. –cpu-max-prime = 1500 and time = 25
    b. –cpu-max-prime = 15000 and time = 25

c. –cpu-max-prime = 150000 and time = 25

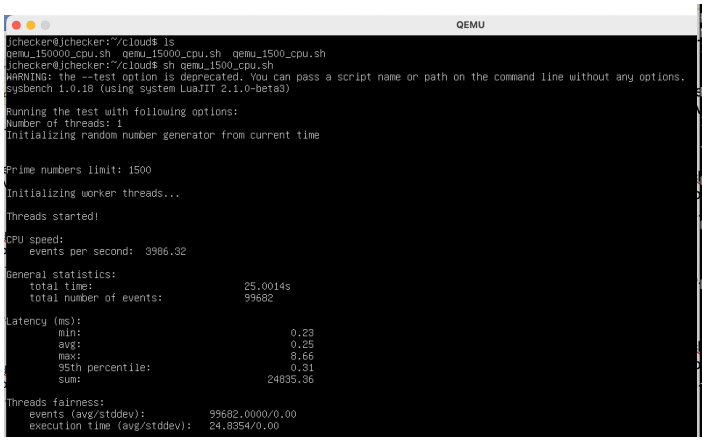
2. FileIO mode
    a. Sequential write (SEQWR)
    b. Combined random Read/Write (RNDRW)

<u>Configuration 1: Running on 3GB memory and 1 core</u>

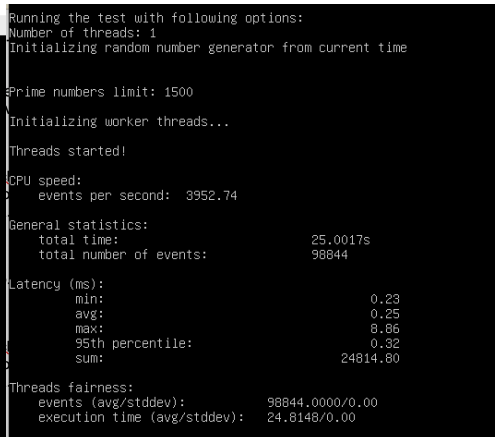
<u>QEMU</u>

juhichecker$ sudo qemu-system-x86_64 -hda ubuntu.img -boot c -m 3G -smp 1 -boot strict=on

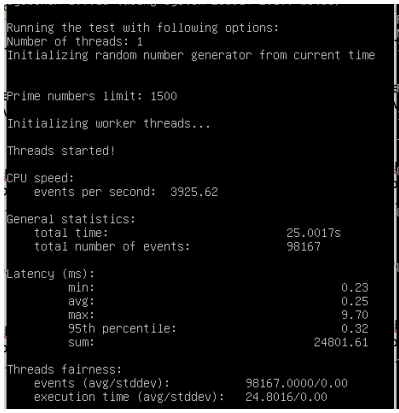
a. CPU Mode
    i. –cpu-max-prime = 1500 and time = 25
    First, I start with a small limit of maximum prime value of 1500 for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

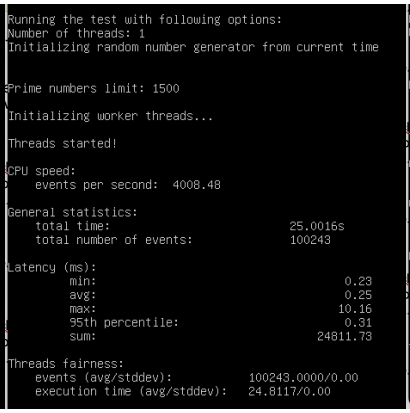


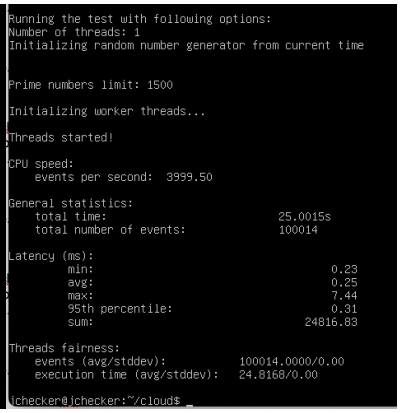
Iteration 1



Iteration 2





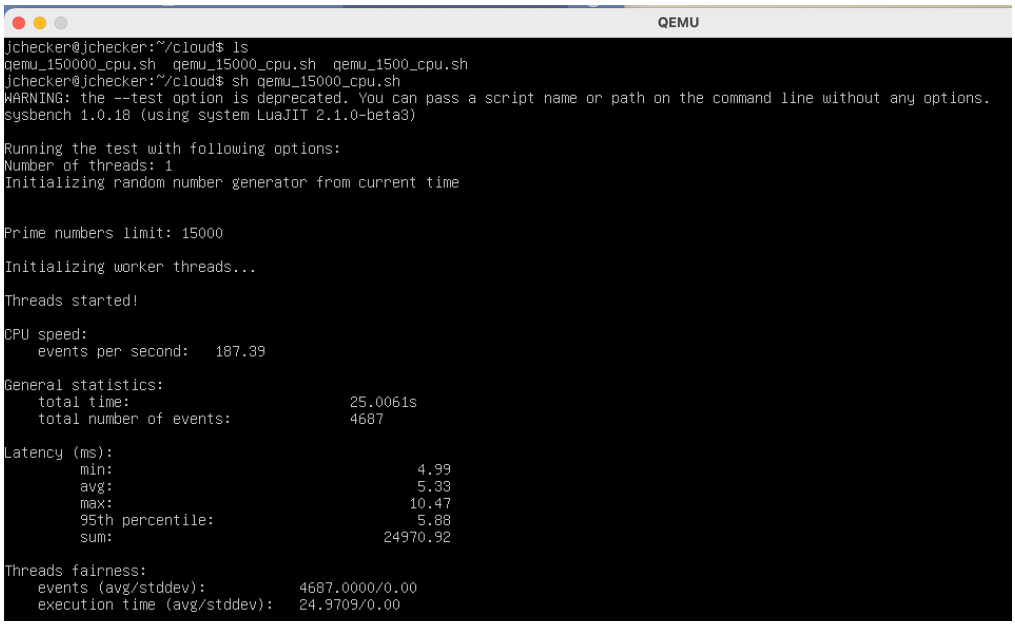


Iteration 3 Iteration 4 Iteration 5

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 0.23 Max = 8.66 Avg = 0.25	3986
2	Min = 0.23 Max = 8.86 Avg =0.25	3952.74
3	Min = 0.23 Max = 9.70 Avg =0.25	3999.5
4	Min = 0.23 Max = 10.16 Avg =0.25	4008.48
5	Min = 0.23 Max = 7.44 Avg =0.25	3925.62

ii. –cpu-max-prime = 15000 and time = 25
I start with a small limit of maximum prime value of 15000 which is 10 times the previous one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

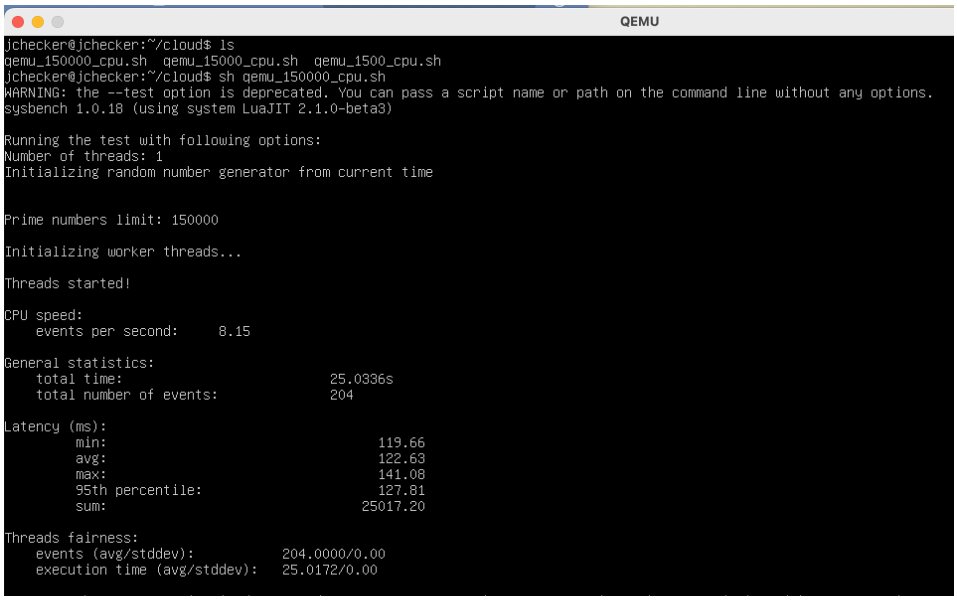


Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 4.99 Max = 10.47 Avg = 5.33	187.79
2	Min = 5.02 Max = 9.65 Avg =5.36	186.16
3	Min = 4.99 Max = 10.50 Avg =5.33	186.14
4	Min = 4.99 Max = 17.76 Avg =5.34	187
5	Min = 5.03 Max = 13.93 Avg =5.36	187.27

iii. –cpu-max-prime = 150000 and time = 25
I start with a small limit of maximum prime value of 150000 which is 100 times the first one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

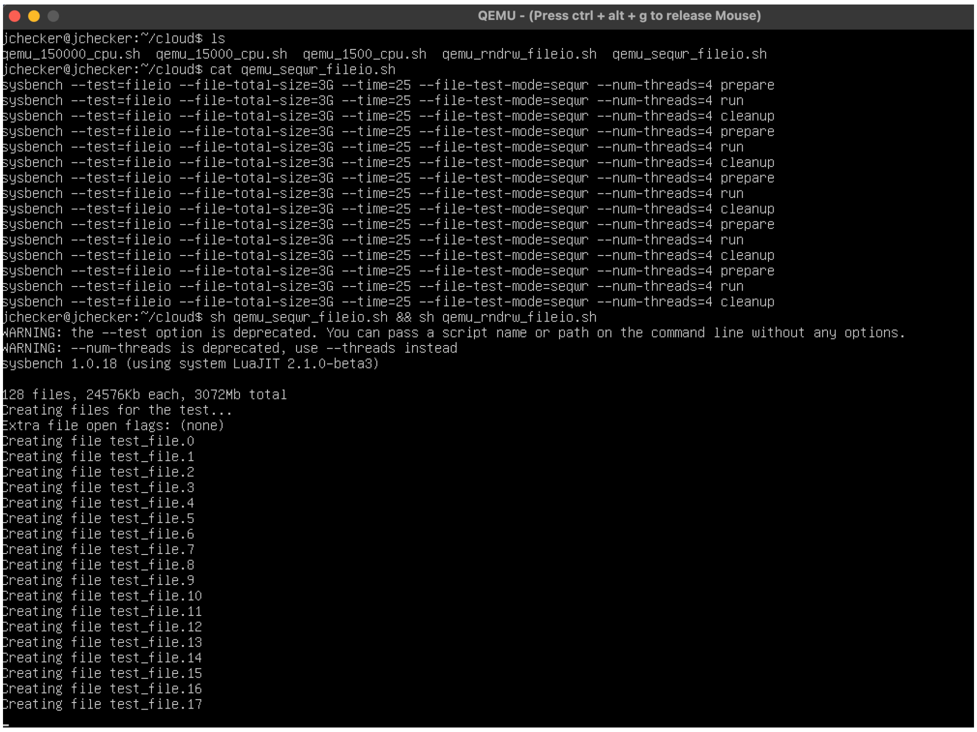


Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 119.66 Max = 141.08 Avg = 122.63	8.15
2	Min = 120.10 Max = 265.79 Avg =129.26	8.04
3	Min = 119.29 Max = 380.16 Avg =133.98	8.04
4	Min = 119.46 Max = 157.34 Avg =123.59	7.44
5	Min = 119.79 Max = 154.15 Avg =124.33	7.22

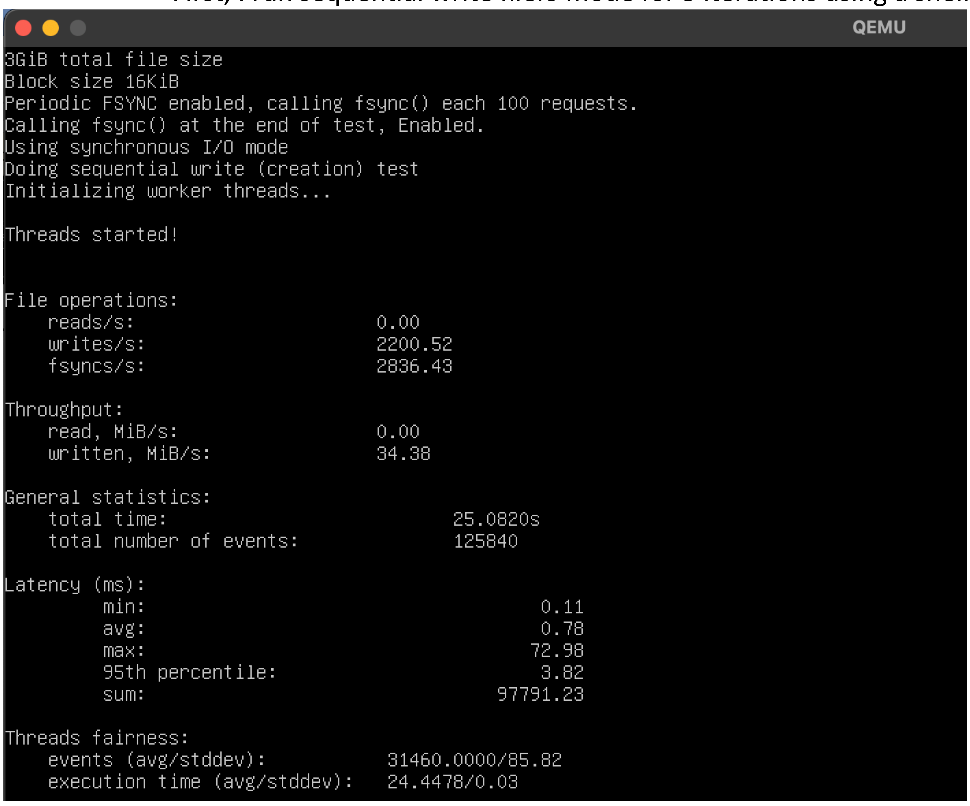
b. Fileio



```
QEMU - (Press ctrl + alt + g to release Mouse)
jchecker@jchecker:~/cloud$ ls
qemu_150000_cpu.sh  qemu_15000_cpu.sh  qemu_1500_cpu.sh  qemu_rndrw_fileio.sh  qemu_seqwr_fileio.sh
jchecker@jchecker:~/cloud$ cat qemu_seqwr_fileio.sh
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 prepare
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 run
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 cleanup
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 prepare
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 run
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 cleanup
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 prepare
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 run
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 cleanup
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 prepare
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 run
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 cleanup
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 prepare
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 run
sysbench --test=fileio --file-total-size=3G --time=25 --file-test-mode=seqwr --num-threads=4 cleanup
jchecker@jchecker:~/cloud$ sh qemu_seqwr_fileio.sh && sh qemu_rndrw_fileio.sh
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

128 files, 24576Kb each, 3072Mb total
Creating files for the test...
Extra file open flags: (none)
Creating file test_file.0
Creating file test_file.1
Creating file test_file.2
Creating file test_file.3
Creating file test_file.4
Creating file test_file.5
Creating file test_file.6
Creating file test_file.7
Creating file test_file.8
Creating file test_file.9
Creating file test_file.10
Creating file test_file.11
Creating file test_file.12
Creating file test_file.13
Creating file test_file.14
Creating file test_file.15
Creating file test_file.16
Creating file test_file.17
```

i. SEQWR

First, I run sequential write fileio mode for 5 iterations using a shell script.



```
QEMU
3GiB total file size
Block size 16KiB
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing sequential write (creation) test
Initializing worker threads...

Threads started!

File operations:
    reads/s:                      0.00
    writes/s:                     2200.52
    fsyncs/s:                     2836.43

Throughput:
    read, MiB/s:                  0.00
    written, MiB/s:               34.38

General statistics:
    total time:                          25.0820s
    total number of events:              125840

Latency (ms):
         min:                                    0.11
         avg:                                    0.78
         max:                                   72.98
         95th percentile:                        3.82
         sum:                                97791.23

Threads fairness:
    events (avg/stddev):           31460.0000/85.82
    execution time (avg/stddev):   24.4478/0.03
```

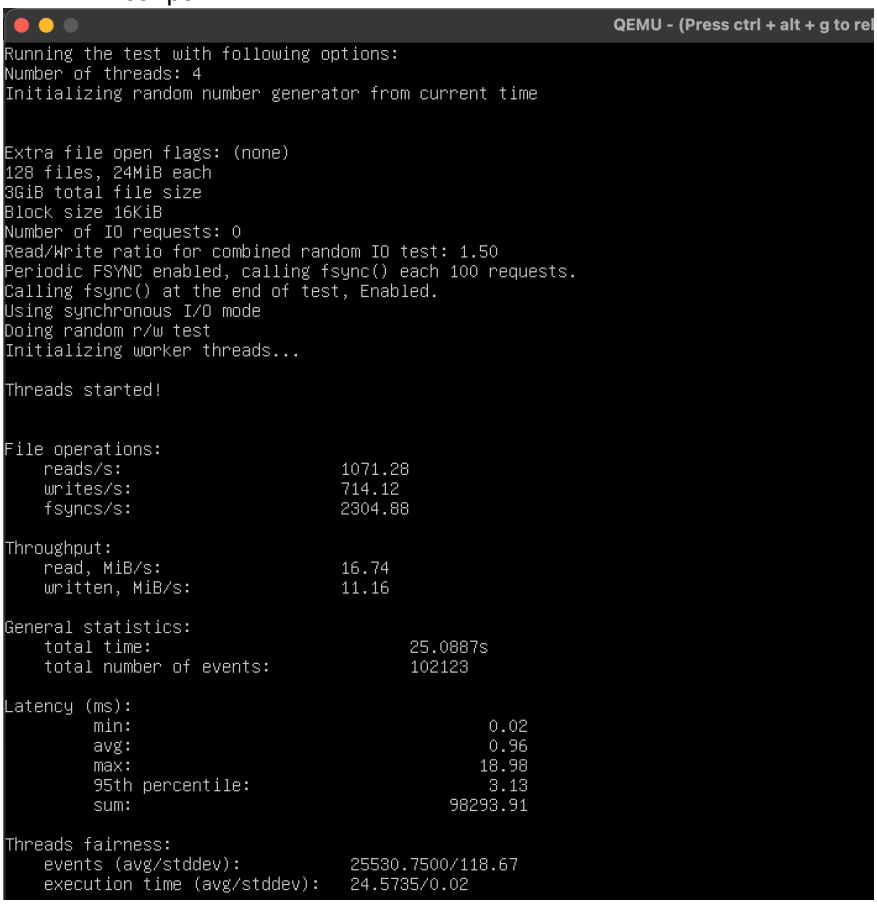
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput (read (Mib/s) = 0.0)
1	Min = 0.11 Max = 72.98 Avg = 0.78	Written (Mib/s) = 34.38
2	Min = 0.11 Max = 39.53 Avg =0.77	Written (Mib/s) = 34.44
3	Min = 0.11 Max = 43.27 Avg =0.78	Written (Mib/s) =31.98
4	Min = 0.11 Max = 84.71 Avg =0.78	Written (Mib/s) =34.8
5	Min = 0.11 Max = 37.02 Avg =0.76	Written (Mib/s) =35.67

ii. RNDRW

I run combined random read write fileio mode for 5 iterations using a shell script.



Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput
1	Min = 0.02 Max = 18.98 Avg = 0.96	Read (Mib/s) = 16.74 Written (Mib/s) = 11.36
2	Min = 0.02 Max = 62.36 Avg = 0.98	Read (Mib/s) =16.87 Written (Mib/s) =11.25
3	Min = 0.02 Max = 21.85 Avg =0.96	Read (Mib/s) =16.87 Written (Mib/s) =10.34
4	Min = 0.02 Max = 23.65 Avg =0.95	Read (Mib/s) =16.47 Written (Mib/s) =11.4
5	Min = 0.02 Max = 19.56 Avg =0.95	Read (Mib/s) =17.23 Written (Mib/s) =1133

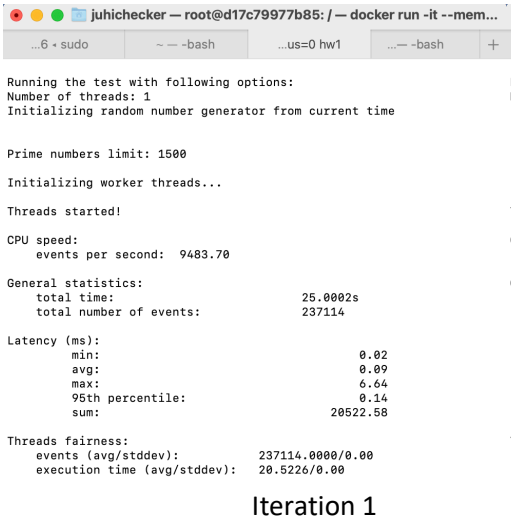
Docker

juhichecker$ docker run -it --memory=3G --cpuset-cpus=0 hw1

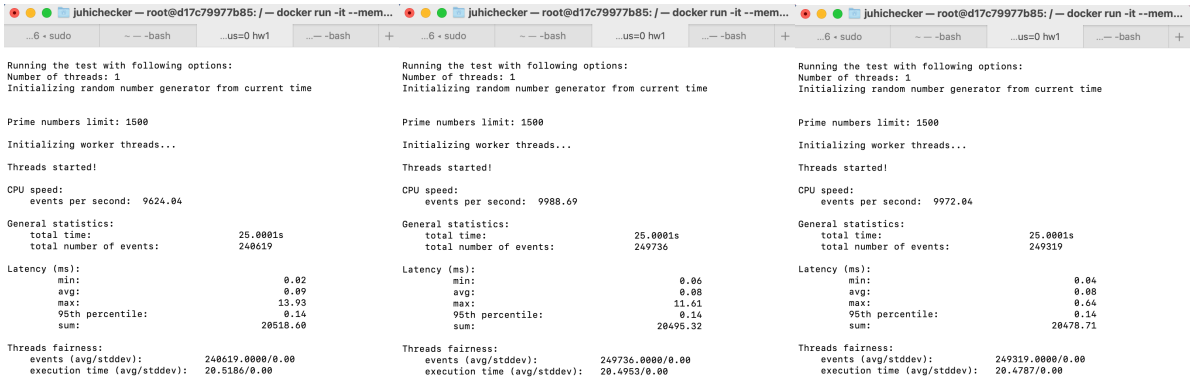
a. CPU Mode

i. –cpu-max-prime = 1500 and time = 25

I start with a small limit of maximum prime value of 1500 for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



Iteration 1 Iteration 2



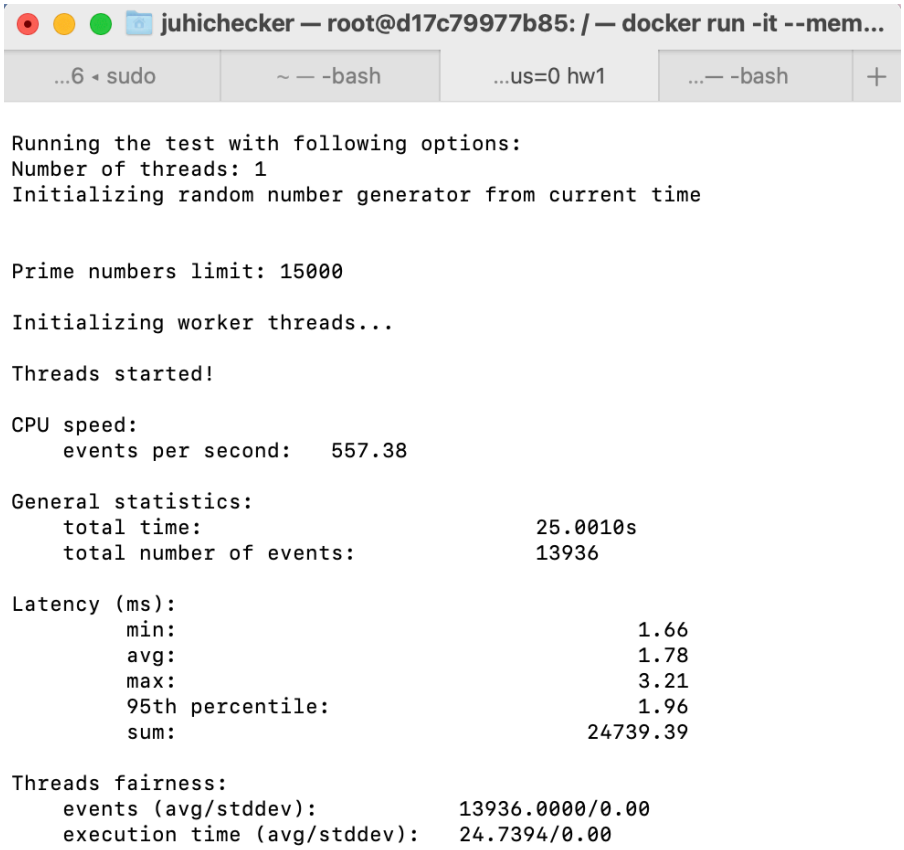
Iteration 1 Iteration 2 Iteration 3

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 0.02 Max = 6.64 Avg = 0.09	9483.70
2	Min = 0.04 Max = 0.64 Avg =0.08	9875.45
3	Min = 0.06 Max = 11.61 Avg =0.08	9624.04
4	Min = 0.06 Max = 4.22 Avg =0.08	9988.0
5	Min = 0.02 Max = 13.93 Avg =0.09	9972.04

ii. –cpu-max-prime = 15000 and time = 25

I start with a small limit of maximum prime value of 15000 which is 10 times the first one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



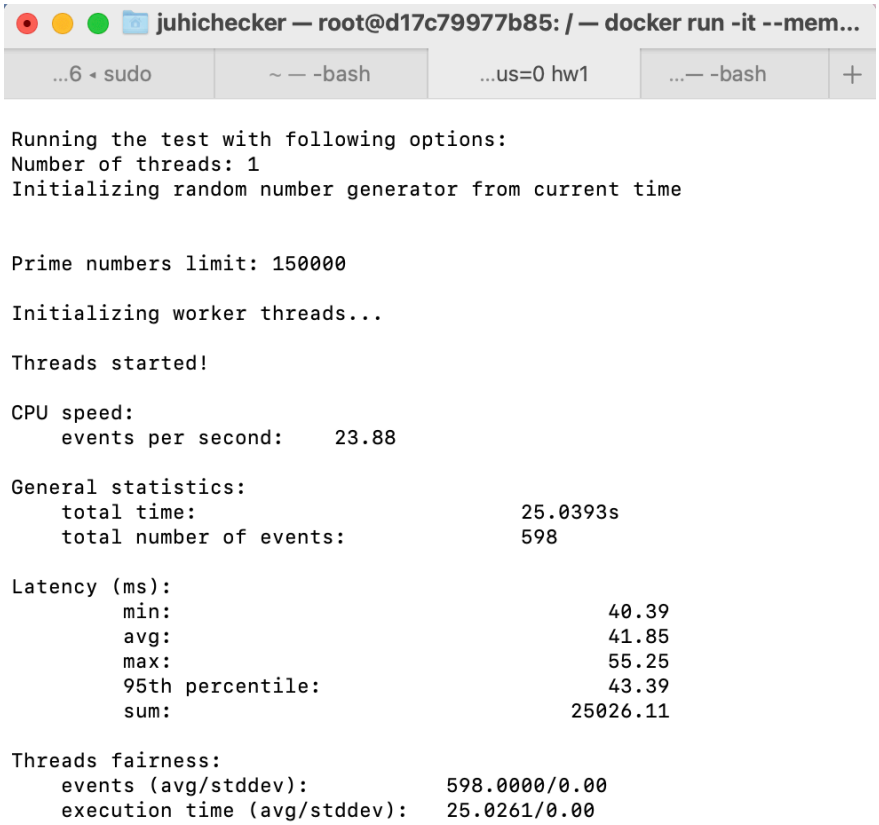
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 1.66 Max = 12.69 Avg = 1.77	557.3
2	Min = 1.66 Max = 3.21 Avg =1.778	567.45
3	Min = 1.66 Max = 6.51 Avg =1.77	524.9
4	Min = 1.66 Max = 3.77 Avg =1.77	558.31
5	Min = 1.66 Max = 3.14 Avg =1.77	553.78

iii. –cpu-max-prime = 150000 and time = 25

I start with a small limit of maximum prime value of 150000 which is 100 times the first one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

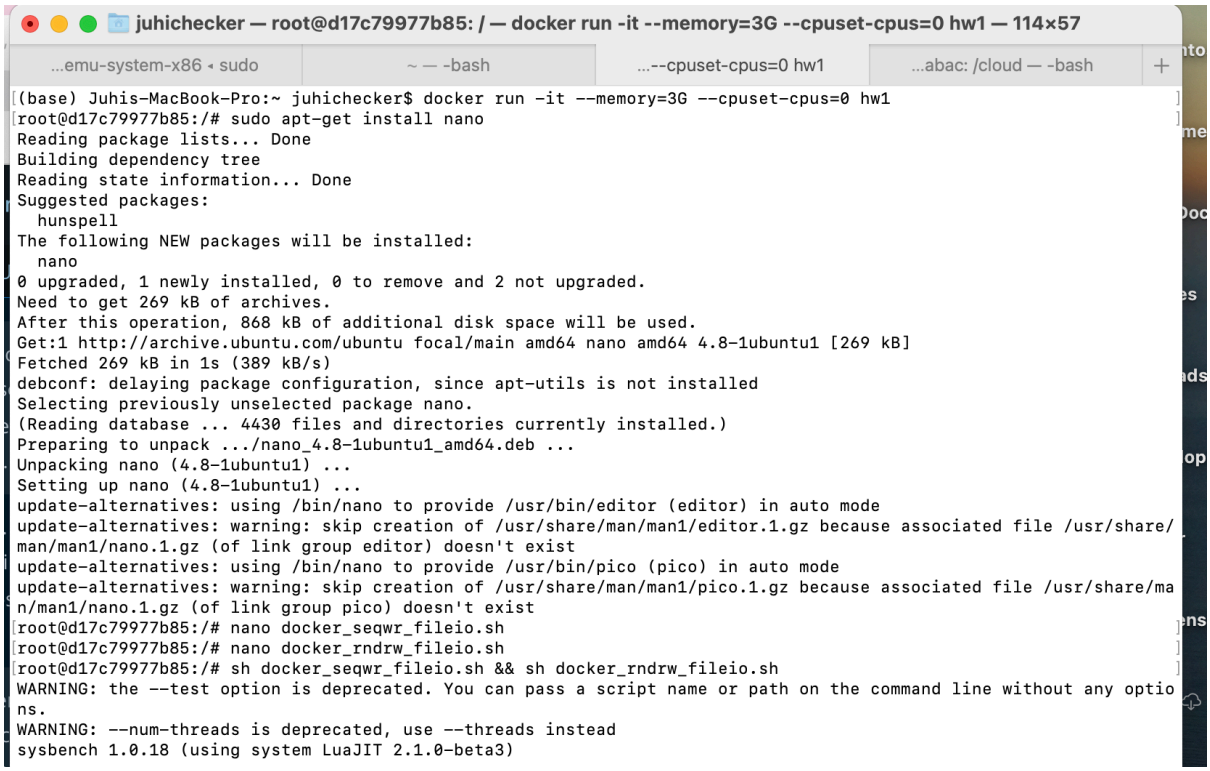


Iteration 1

The table below are the latency scores after the 5 iterations:

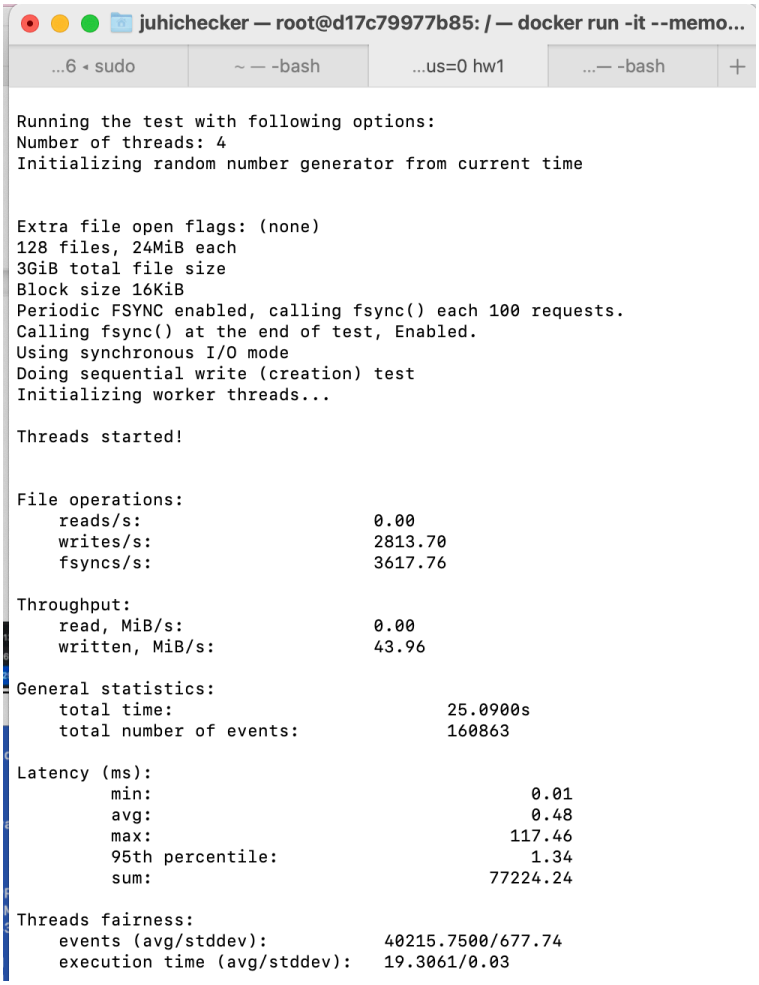
Iterations	Latency	Events/sec
1	Min = 40.39 Max = 55.25 Avg = 41.05	23.88
2	Min = 40.89 Max = 55.99 Avg = 41.96	23.97
3	Min = 40.9 Max = 51.12 Avg =41.92	29.35
4	Min = 40.88 Max = 49.0 Avg =41.78	23.78
5	Min = 40.66 Max = 50.67 Avg =41.9	22.85

b. FileIO



i. SEQWR

I run sequential write fileio mode for 5 iterations using a shell script.



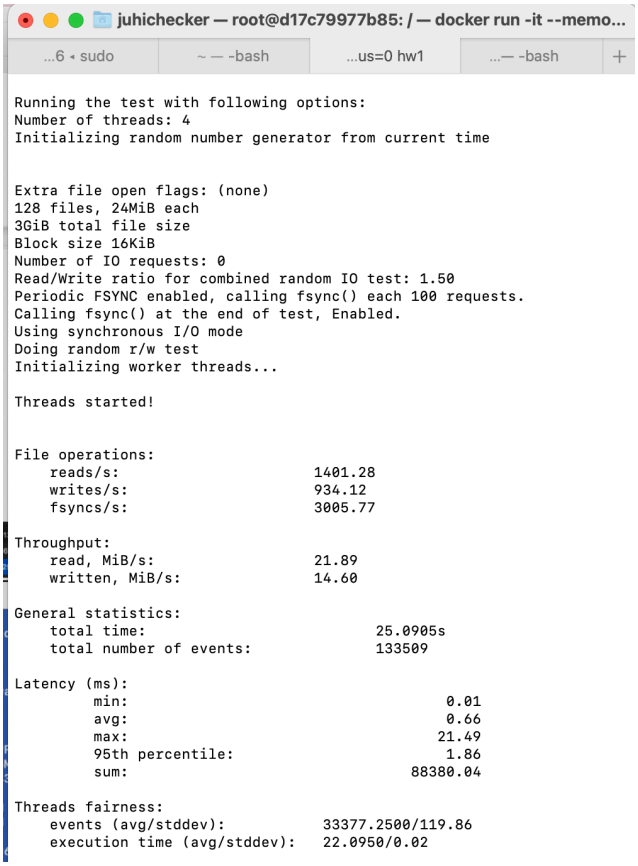
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput (read (Mib/s) = 0.0)
1	Min = 0.01 Max = 170.32 Avg = 0.62	Written (Mib/s) = 43.96
2	Min = 0.01 Max = 379.22 Avg =0.97	Written (Mib/s) = 39.05
3	Min = 0.01 Max = 84.11 Avg =0.55	Written (Mib/s) =37.45
4	Min = 0.01 Max = 117.46 Avg =0.48	Written (Mib/s) =36.89
5	Min = 0.01 Max = 127.63 Avg =0.58	Written (Mib/s) =39.35

ii. RNDRW

I run combined random read write fileio mode for 5 iterations using a shell script.



Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput
1	Min = 0.01 Max = 21.49 Avg = 0.66	Read (Mib/s) = 21.89 Written (Mib/s) = 14.60
2	Min = 0.01 Max = 186.20 Avg =0.72	Read (Mib/s) =28.53 Written (Mib/s) =13.21
3	Min = 0.01 Max = 154.36 Avg =0.70	Read (Mib/s) =21.45 Written (Mib/s) =14.34
4	Min = 0.01 Max = 45.43 Avg =0.68	Read (Mib/s) =21.78 Written (Mib/s) =14.24
5	Min = 0.01 Max = 42.49 Avg =0.68	Read (Mib/s) =21.34 Written (Mib/s) =14.34

Conclusion:

1. **In CPU mode test it is observed that as the number of max prime value increases the latency of QEMU and Docker also increases. However the latency of QEMU is higher than that of Docker for almost all the cases.**
2. **In File IO mode test it is observed that for sequential write operation latency of QEMU was almost similar to that of Docker. In combined random read write operation the latency for docker was less than that of QEMU.**

Configuration 2: Running on 6GB memory and 2 core

QEMU

juhichecker$ sudo qemu-system-x86_64 -hda ubuntu.img -boot c -m 6G -smp 2 -boot strict=on

a. CPU Mode
   i. –cpu-max-prime = 1500 and time = 25
   First, I start with a small limit of maximum prime value of 1500 for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

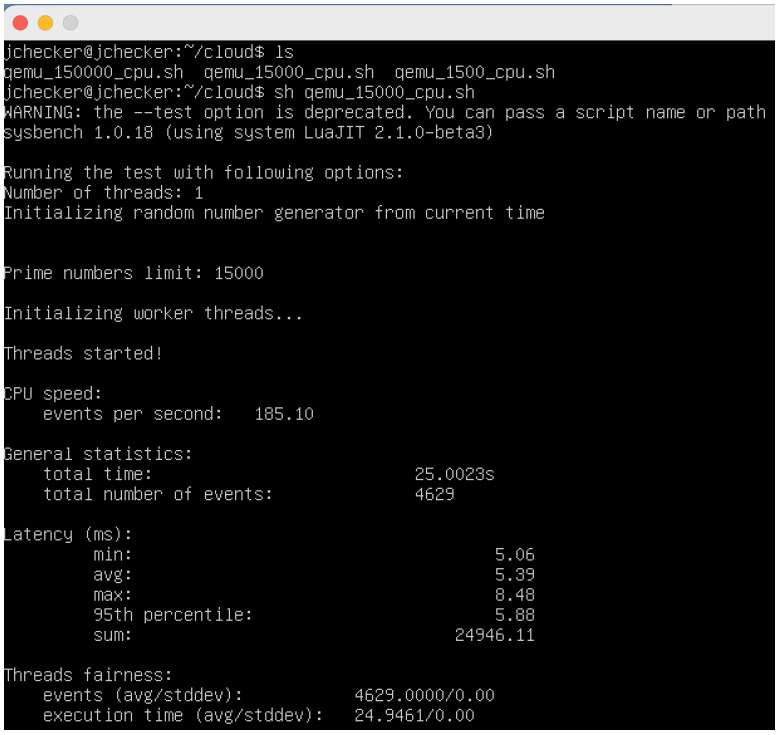
```
jchecker@jchecker:~$ cd cloud/ && ls
qemu_150000_cpu.sh  qemu_15000_cpu.sh  qemu_1500_cpu.sh
jchecker@jchecker:~/cloud$ sh qemu_1500_cpu.sh
WARNING: the --test option is deprecated. You can pass a script name or path
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 1500

Initializing worker threads...

Threads started!

CPU speed:
    events per second:  3917.98

General statistics:
    total time:                          25.0016s
    total number of events:              97979

Latency (ms):
         min:                                    0.23
         avg:                                    0.25
         max:                                    2.16
         95th percentile:                        0.32
         sum:                                24560.92

Threads fairness:
    events (avg/stddev):           97979.0000/0.00
    execution time (avg/stddev):   24.5609/0.00
```

Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 0.23 Max = 2.16 Avg = 0.25	3917.98
2	Min = 0.23 Max = 3.10 Avg =0.25	3889.99
3	Min = 0.23 Max = 4.11 Avg =0.25	3908.13
4	Min = 0.23 Max = 3.91 Avg =0.25	3899.13
5	Min = 0.23 Max = 3.51 Avg =0.25	3945.14

ii. –cpu-max-prime = 15000 and time = 25
I start with a small limit of maximum prime value of 15000 which is 10 times the previous one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

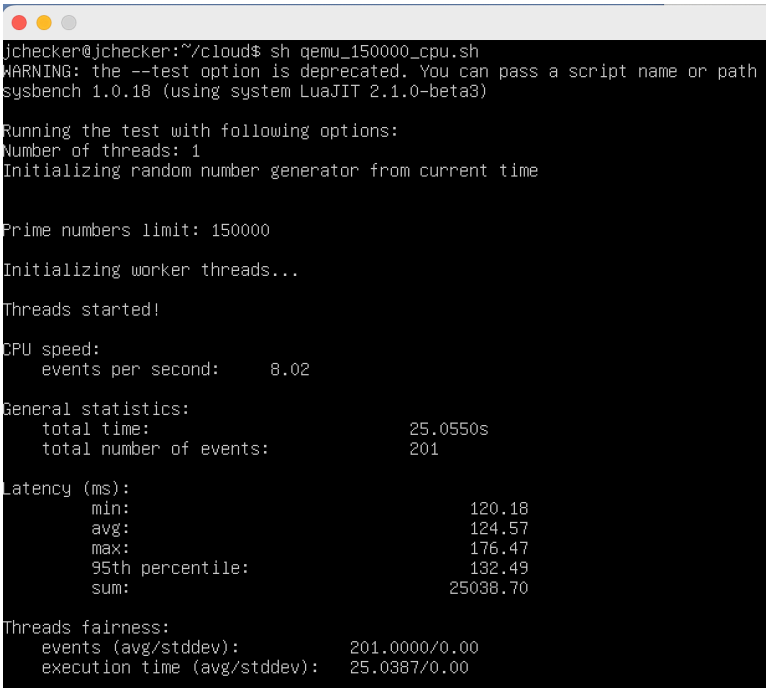


Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 5.06 Max = 8.48 Avg = 5.39	185.1
2	Min = 4.99 Max = 9.53 Avg =5.33	186.34
3	Min = 4.99 Max = 10.31 Avg =5.37	187.19
4	Min = 5.01 Max = 9.92 Avg = 5.34	185.34
5	Min = 5.02 Max = 10.28 Avg =5.35	189.23

iii. –cpu-max-prime = 150000 and time = 25
I start with a small limit of maximum prime value of 150000 which is 100 times the first one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



Iteration 1

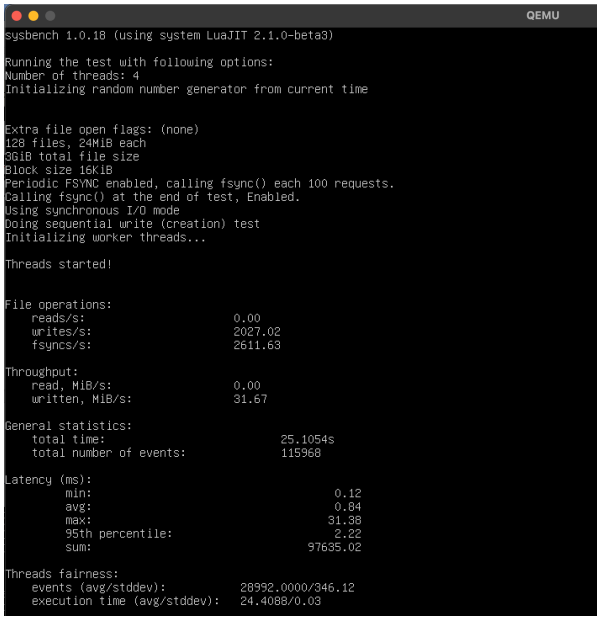
The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 120.18 Max = 176.47 Avg = 124.12	8.02
2	Min = 119.9 Max = 174.1 Avg =124.66	7.99
3	Min = 120.4 Max = 175.8 Avg =125.1	8.01
4	Min = 119.9 Max = 152.2 Avg =123.9	7.78
5	Min = 120.5 Max = 170.0 Avg =125.4	8.06

b. FileIO

i. SEQWR

First, I run sequential write fileio mode for 5 iterations using a shell script.



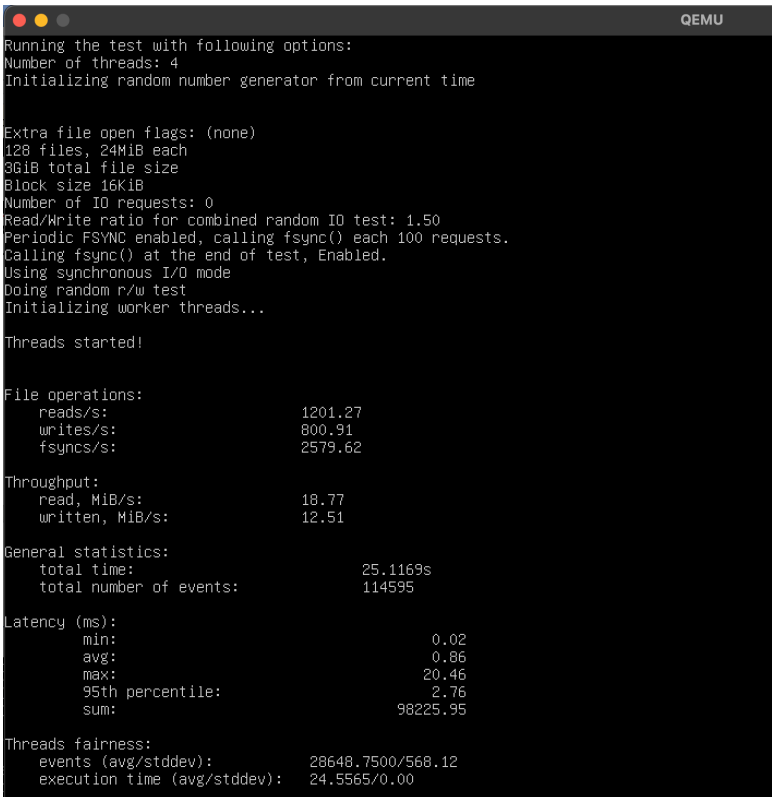
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput (read (Mib/s) = 0.0)
1	Min = 0.12 Max = 31.38 Avg = 0.84	Written (Mib/s) = 31.67
2	Min = 0.17 Max = 83.3 Avg =0.85	Written (Mib/s) = 31.05
3	Min = 0.12 Max = 78.16 Avg =0.9	Written (Mib/s) =35.45
4	Min = 0.12 Max = 70.38 Avg =0.85	Written (Mib/s) =29.35
5	Min = 0.12 Max = 35.1 Avg =0.84	Written (Mib/s) =31.35

ii. RNDRW

I run combined random read write fileio mode for 5 iterations using a shell script.



Iteration 1

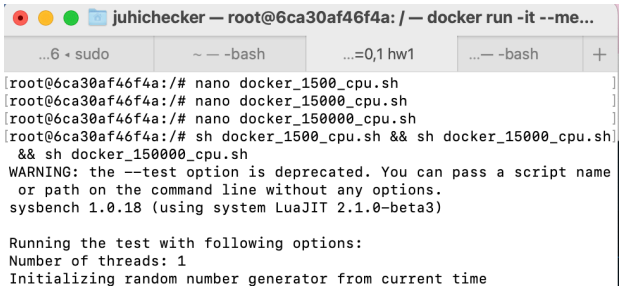
The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput
1	Min = 0.02 Max = 20.4 Avg = 0.86	Read (Mib/s) = 18.77 Written (Mib/s) = 12.51
2	Min = 0.02 Max = 26.3 Avg =0.84	Read (Mib/s) =19.00 Written (Mib/s) =12.34
3	Min = 0.02 Max = 80.6 Avg =0.86	Read (Mib/s) =18.23 Written (Mib/s) =12.89
4	Min = 0.02 Max = 17.2 Avg =0.85	Read (Mib/s) =19.23 Written (Mib/s) =13.02
5	Min = 0.02 Max = 20.19 Avg =0.84	Read (Mib/s) =19.34 Written (Mib/s) =12.99

Docker

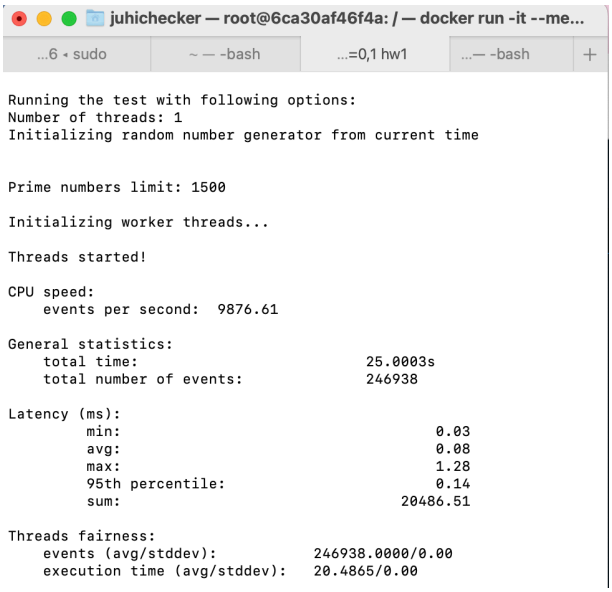
juhichecker$ docker run -it --memory=6G --cpuset-cpus=0,1 hw1

a. CPU Mode



i. –cpu-max-prime = 1500 and time = 25

First, I start with a small limit of maximum prime value of 1500 for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



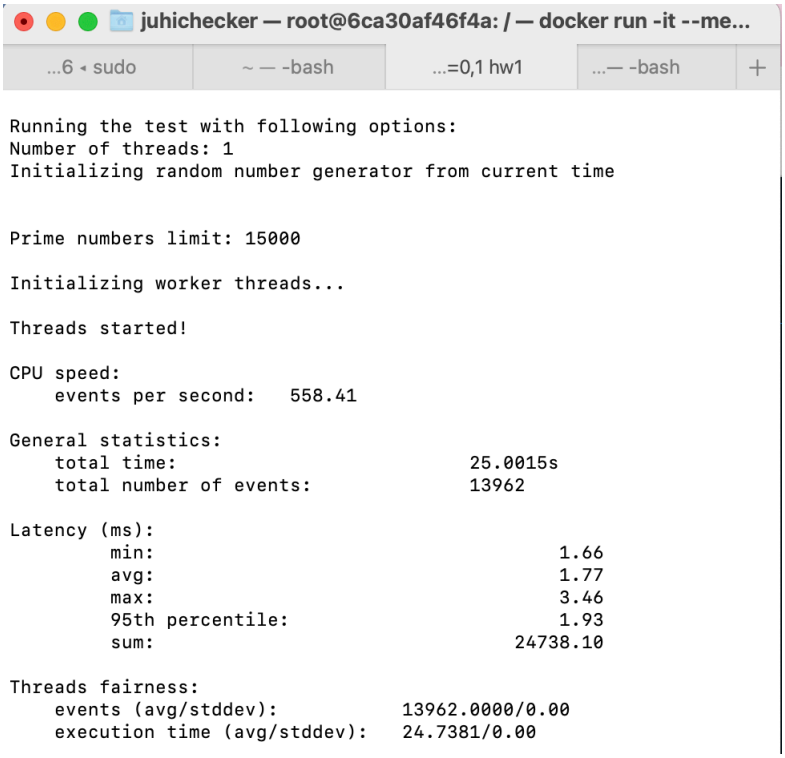
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 0.03 Max = 1.28 Avg = 0.08	9876.34
2	Min = 0.02 Max = 1.45 Avg =0.08	9983.01
3	Min = 0.05 Max = 7.39 Avg =0.08	9934.31
4	Min = 0.06 Max = 0.66 Avg =0.08	9913.90
5	Min = 0.03 Max = 1.06 Avg =0.08	98913.4

ii. –cpu-max-prime = 15000 and time = 25

I start with a small limit of maximum prime value of 15000 which is 10 times the previous one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



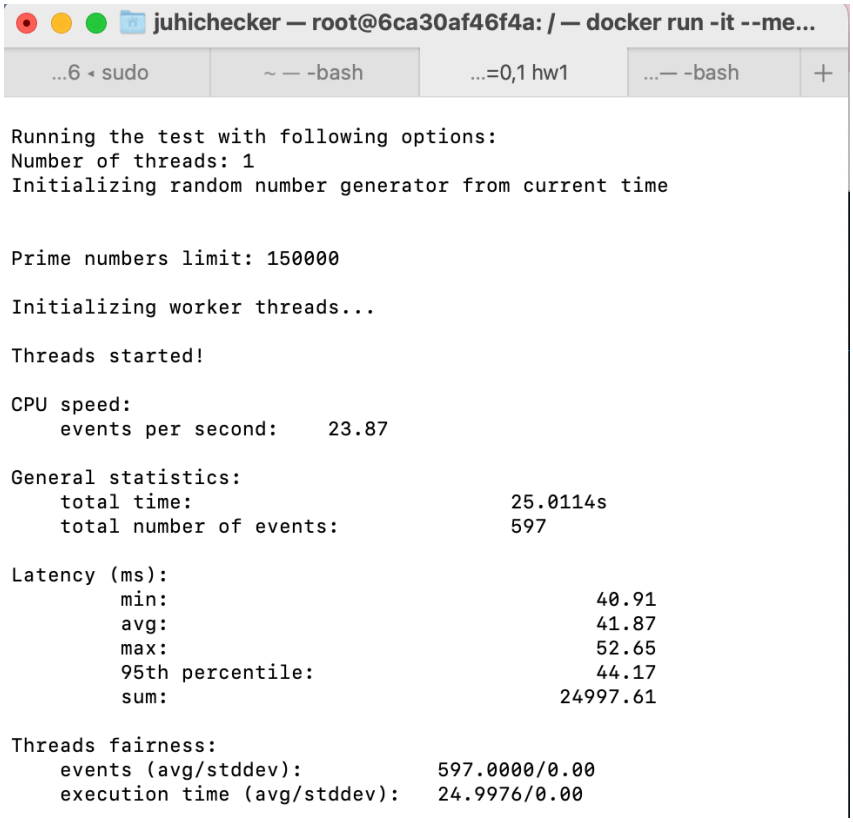
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 1.66 Max = 3.46 Avg = 1.77	558.41
2	Min = 1.62 Max = 3.08 Avg =1.77	557.34
3	Min = 1.66 Max = 3.32 Avg =1.78	560.32
4	Min = 1.66 Max = 3.13 Avg =1.77	555.24
5	Min = 1.66 Max = 3.63 Avg =1.77	576.34

iii. –cpu-max-prime = 150000 and time = 25

I start with a small limit of maximum prime value of 150000 which is 100 times the first one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

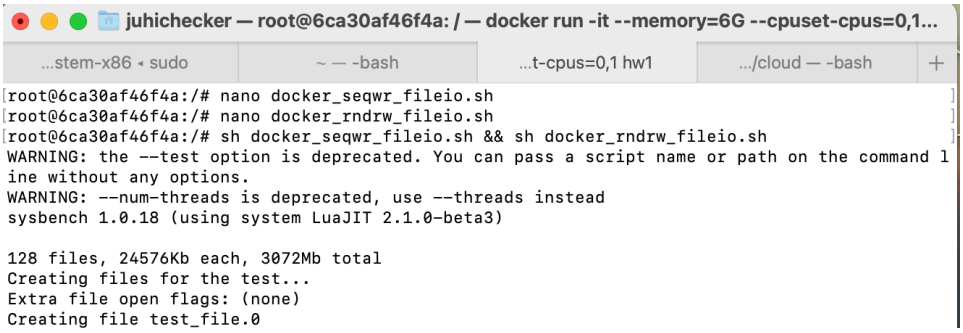


Iteration 1

The table below are the latency scores after the 5 iterations:

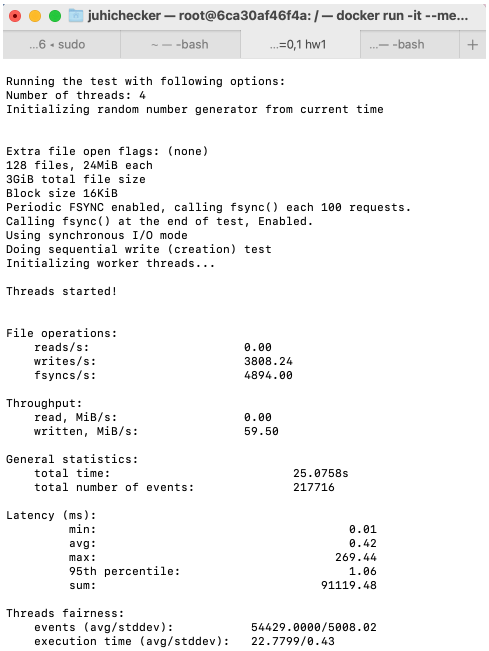
Iterations	Latency	Events/sec
1	Min = 40.91 Max = 52.65 Avg = 41.87	23.87
2	Min = 41.0 Max = 50.3 Avg =42.83	22.34
3	Min = 41.01 Max = 51.47 Avg =42.63	22.50
4	Min = 40.98 Max = 53.86 Avg =42.49	23.45
5	Min = 41.0 Max = 57.4 Avg =42.69	22.34

b. FileIO



i. SEQWR

First, I run sequential write fileio mode for 5 iterations using a shell script.



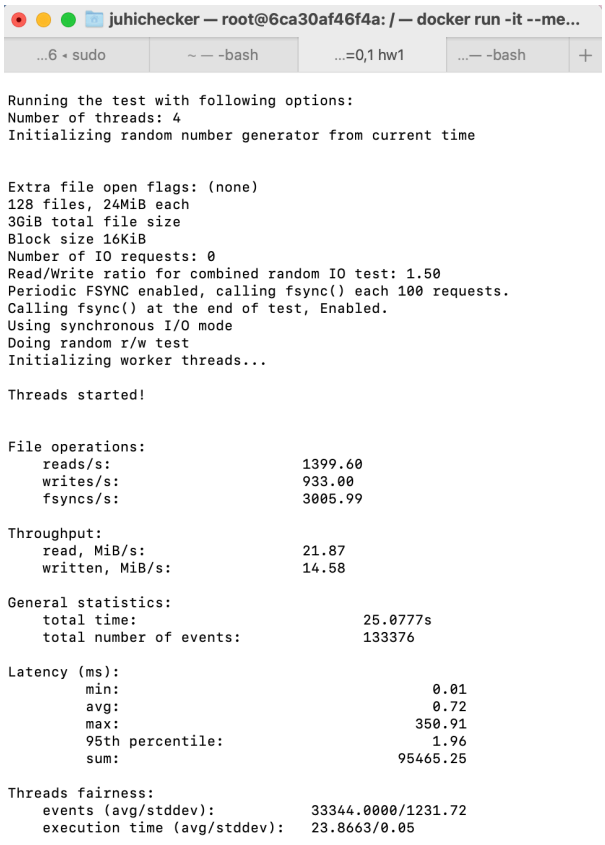
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput (read (Mib/s) = 0.0)
1	Min = 0.01 Max = 269.44 Avg = 0.42	Written (Mib/s) = 59.89
2	Min = 0.01 Max = 347.3 Avg =0.42	Written (Mib/s) = 58.67
3	Min = 0.01 Max = 465.7 Avg =0.43	Written (Mib/s) =59.03
4	Min = 0.01 Max = 354.94 Avg =0.47	Written (Mib/s) =58.32
5	Min = 0.01 Max = 363.1 Avg =0.47	Written (Mib/s) =58.88

ii. RNDRW

I run combined random read write fileio mode for 5 iterations using a shell script.



Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput
1	Min = 0.01 Max = 350.9 Avg = 0.72	Read (Mib/s) = 21.87 Written (Mib/s) = 14.58
2	Min = 0.00 Max = 255.52 Avg =0.67	Read (Mib/s) =22.34 Written (Mib/s) =15.01
3	Min = 0.0 Max = 434.43 Avg =0.70	Read (Mib/s) =23.21 Written (Mib/s) =14.99
4	Min = 0.01 Max = 112.39 Avg =0.66	Read (Mib/s) =21.34 Written (Mib/s) =15.34
5	Min = 0.01 Max = 203.63 Avg =0.64	Read (Mib/s) =24.45 Written (Mib/s) =15.34

Conclusion:

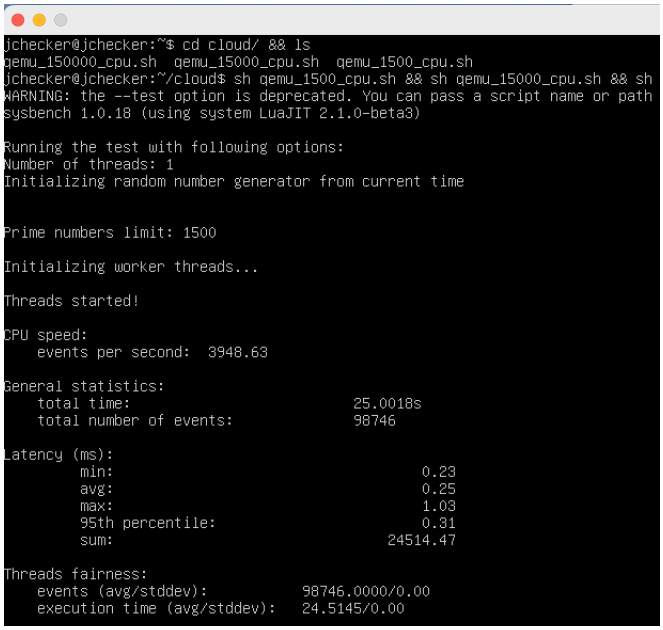
1. **In CPU mode test it is observed that as the number of max prime value increases the latency of QEMU and Docker also increases. It is observed that on smaller tasks and very huge tasks QEMU's latency is better than Docker, whereas on average tasks Docker latency is slightly better than QEMU.**
2. **In File IO mode test it is observed that for sequential write operation latency of QEMU was less than that of Docker. In combined random read write operation the latency for docker was less than that of QEMU.**
3. **There was no significant changes seen on doubling the memory and number of cores.**

Configuration 3: Running on 9GB memory and 4 core

QEMU

juhichecker$ sudo qemu-system-x86_64 -hda ubuntu.img -boot c -m 9G -smp 4 -boot strict=on

a. CPU Mode
    i. –cpu-max-prime = 1500 and time = 25
    First, I start with a small limit of maximum prime value of 1500 for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

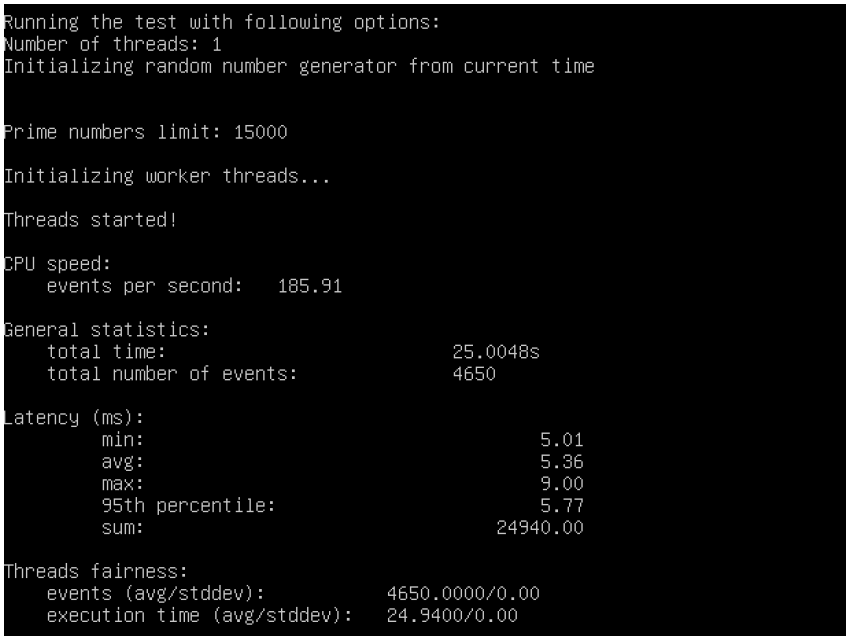


Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 0.23 Max = 1.03 Avg = 0.25	3948.63
2	Min = 0.23 Max = 2.55 Avg =0.25	3984.34
3	Min = 0.23 Max = 2.52 Avg =0.25	4004.24
4	Min = 0.23 Max = 2.46 Avg =0.25	3986.345
5	Min = 0.23 Max = 2.92 Avg =0.25	3899.34

ii. –cpu-max-prime = 15000 and time = 25
I start with a small limit of maximum prime value of 15000 which is 10 times the previous one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 5.01 Max = 9.0 Avg = 5.36	185.91
2	Min = 5.03 Max = 9.14 Avg =5.36	187.34
3	Min = 5.01 Max = 8.77 Avg =5.33	188.34
4	Min = 4.99 Max = 8.557 Avg =5.32	187.0
5	Min = 4.9 Max = 10.01 Avg =5.33	189.9

iii. –cpu-max-prime = 150000 and time = 25
I start with a small limit of maximum prime value of 150000 which is 100 times the first one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

```
Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 150000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:     8.09

General statistics:
    total time:                          25.0969s
    total number of events:              203

Latency (ms):
         min:                                  120.40
         avg:                                  123.52
         max:                                  154.01
         95th percentile:                      130.13
         sum:                                25075.23

Threads fairness:
    events (avg/stddev):           203.0000/0.00
    execution time (avg/stddev):   25.0752/0.00
```

Iteration 1

The table below are the latency scores after the 5 iterations:

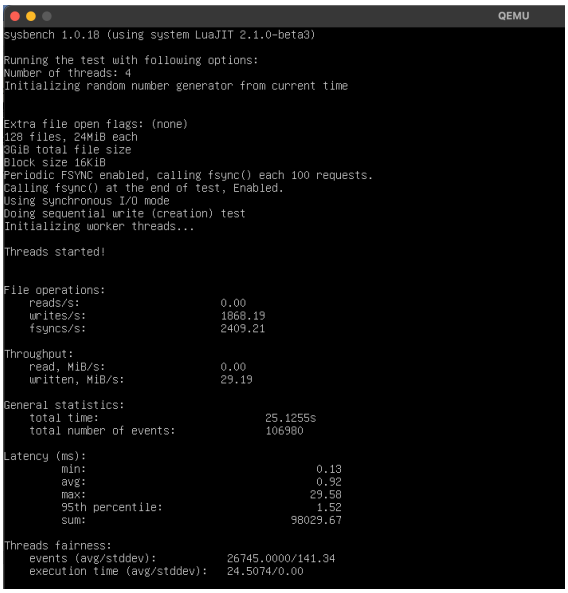
Iterations	Latency	Events/sec
1	Min = 120.99 Max = 169.73 Avg = 16.14	8.09
2	Min = 120.4 Max = 154.13 Avg =123.52	8.04
3	Min = 120.84 Max = 153.91 Avg =124.28	8.11
4	Min = 119.75 Max = 142.67 Avg =123.75	7.92
5	Min = 120.25 Max = 156.26 Avg =123.82	8..07

b. FileIO Mode

```
QEMU
jchecker@jchecker:~/cloud$ sh qemu_seqwr_fileio.sh && sh qemu_rndrw_fileio.sh
WARNING: the --test option is deprecated. You can pass a script name or path on the comm
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

128 files, 24576Kb each, 3072Mb total
Creating files for the test...
Extra file open flags: (none)
Creating file test_file.0
Creating file test_file.1
Creating file test_file.2
Creating file test_file.3
Creating file test_file.4
Creating file test_file.5
```

i. SEQWR

First, I run sequential write fileio mode for 5 iterations using a shell script.



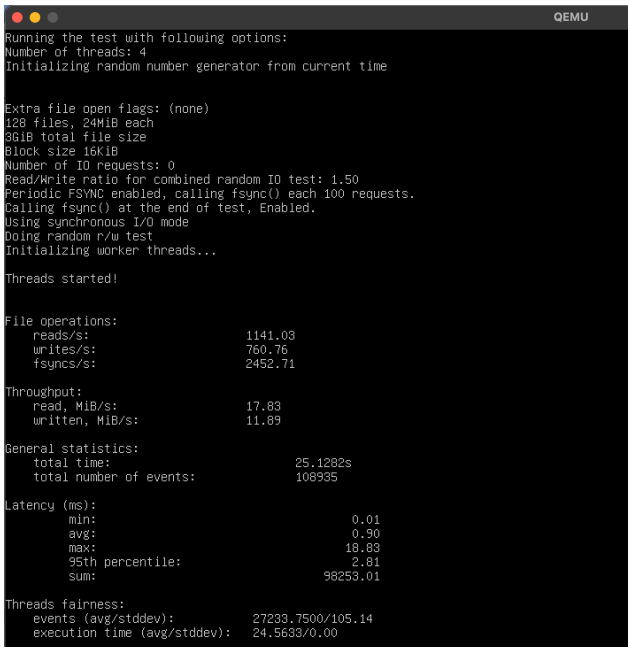
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput (read (Mib/s) = 0.0)
1	Min = 0.13 Max = 29.58 Avg = 0.92	Written (Mib/s) = 29.19
2	Min = 0.68 Max = 51.11 Avg =0.95	Written (Mib/s) = 30.21
3	Min = 0.12 Max = 25.3 Avg =0.91	Written (Mib/s) =29.99
4	Min = 0.13 Max = 26.45 Avg =0.9	Written (Mib/s) =30.38
5	Min = 0.12 Max = 32.68 Avg =0.93	Written (Mib/s) =29.91

ii. RNDRW

I run combined random read write fileio mode for 5 iterations using a shell script.



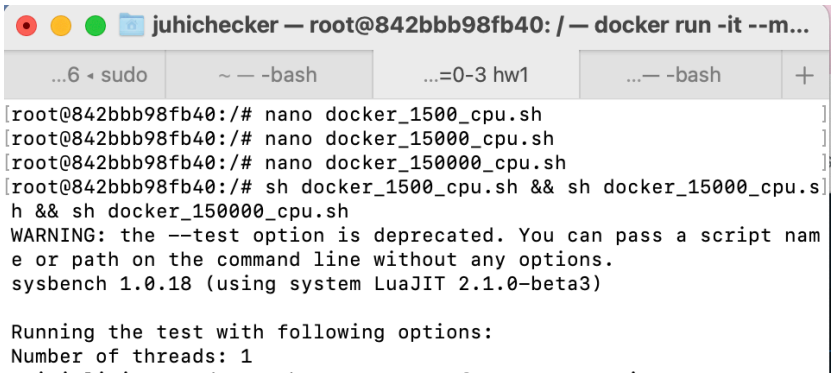
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput
1	Min = 0.01 Max = 119.4 Avg = 0.09	Read (Mib/s) = 17.83 Written (Mib/s) = 11.09
2	Min = 0.01 Max = 18.82 Avg =0.9	Read (Mib/s) =18.34 Written (Mib/s) =10.99
3	Min = 0.01 Max = 35.72 Avg =0.94	Read (Mib/s) =18.3 Written (Mib/s) =11.23
4	Min = 0.0 Max = 17.2 Avg =0.9	Read (Mib/s) =17.23 Written (Mib/s) =12.34
5	Min = 0.01 Max = 17.24 Avg =0.08	Read (Mib/s) =18.98 Written (Mib/s) =11.63

Docker

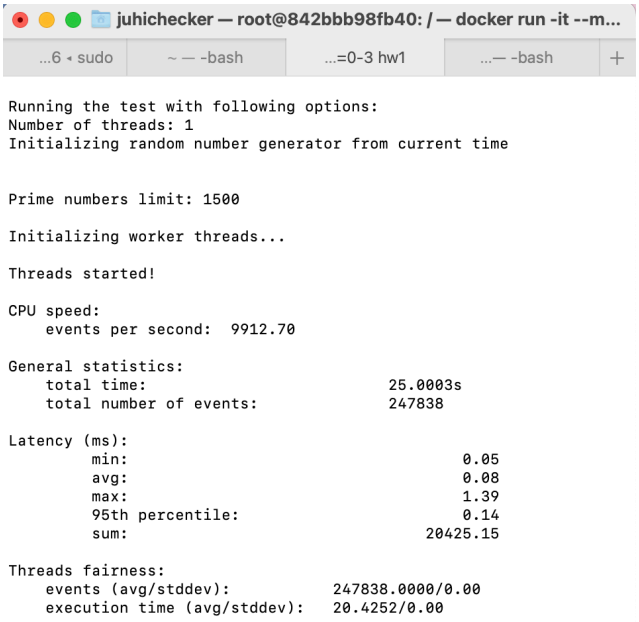
juhichecker$ docker run -it --memory=9G --cpuset-cpus=0-3 hw1



a. CPU Mode

   i. –cpu-max-prime = 1500 and time = 25

   First, I start with a small limit of maximum prime value of 1500 for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



Iteration 1

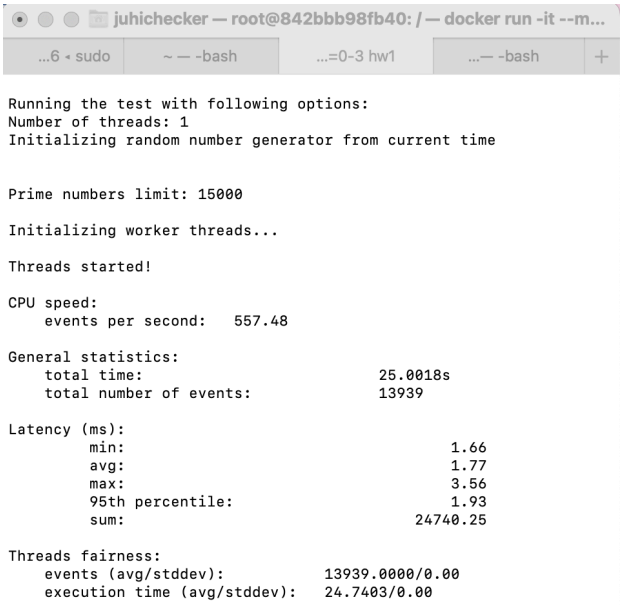
The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 0.04 Max = 1.01 Avg = 0.08	9921.78
2	Min = 0.04 Max = 0.82 Avg =0.08	9934.13
3	Min = 0.01 Max = 2.53 Avg =0.09	9957.56

4	Min = 0.05 Max = 1.39 Avg = 0.08	9744.24
5	Min = 0.06 Max = 0.73 Avg = 0.08	9667.8

ii. –cpu-max-prime = 15000 and time = 25

I start with a small limit of maximum prime value of 15000 which is 10 times the previous one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.



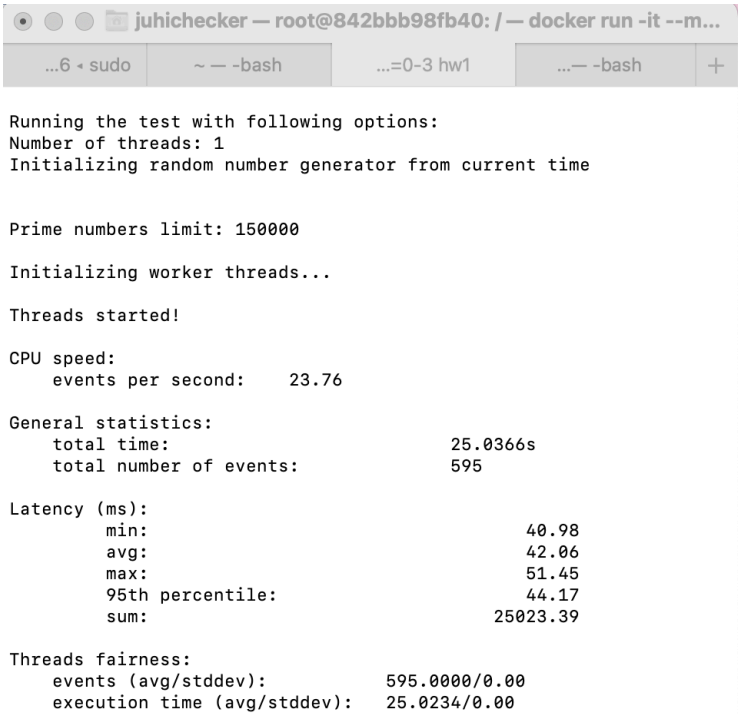
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Events/sec
1	Min = 1.66 Max = 4.99 Avg = 1.80	557.48
2	Min = 1.66 Max = 4.36 Avg = 1.79	546.7
3	Min = 1.66 Max = 4.67 Avg = 1.78	589.42
4	Min = 1.66 Max = 3.56 Avg =1.77	588.24
5	Min = 1.66 Max = 14.15 Avg =1.78	555.3

iii. –cpu-max-prime = 150000 and time = 25

I start with a small limit of maximum prime value of 150000 which is 100 times the first one for a time limit of 25 sec. I run this case for 5 iterations using a shell script.

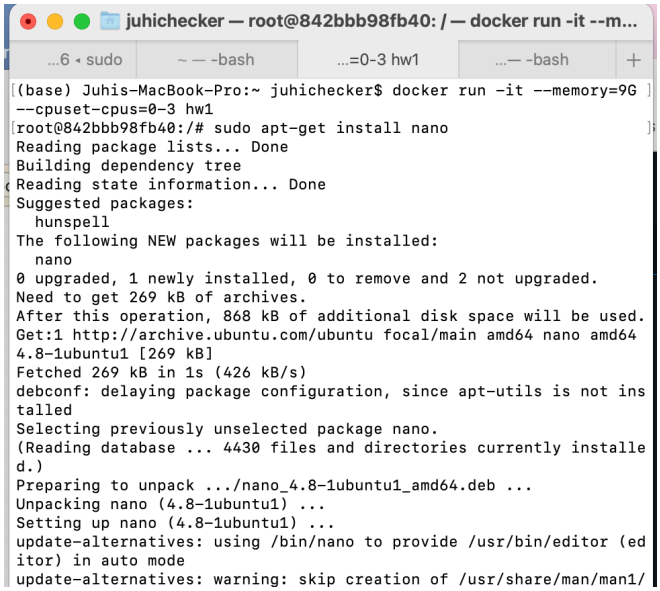


Iteration 1

The table below are the latency scores after the 5 iterations:

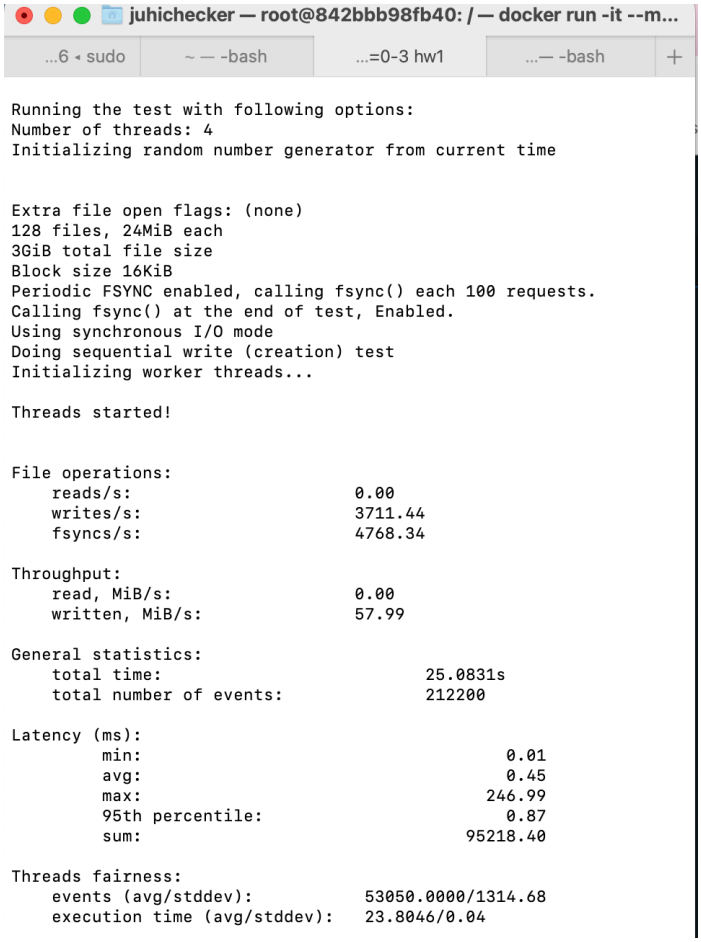
Iterations	Latency	Events/sec
1	Min = 41.02 Max = 51.82 Avg = 41.83	23.81
2	Min = 48.99 Max = 59.50 Avg =41.98	22.45
3	Min = 41.02 Max = 50.81 Avg =41.89	25.24
4	Min = 40.98 Max = 51.45 Avg =42.96	23.23
5	Min = 41.00 Max = 50.89 Avg =41.98	23.45

b. FileIO Mode



i. SEQWR

First, I run sequential write fileio mode for 5 iterations using a shell script.



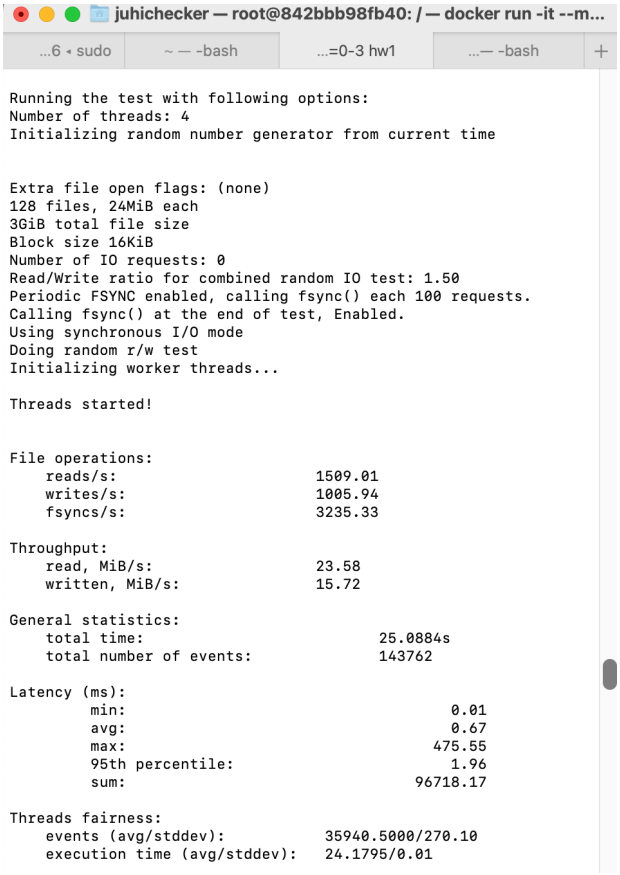
Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput (read (Mib/s) = 0.0)
1	Min = 0.01 Max = 246.99 Avg = 0.45	Written (Mib/s) = 57.99
2	Min = 0.01 Max = 280.4 Avg =0.49	Written (Mib/s) = 55.78
3	Min = 0.01 Max = 321.78 Avg =0.47	Written (Mib/s) =54.67
4	Min = 0.01 Max = 263.55 Avg =0.46	Written (Mib/s) =55.67
5	Min = 0.01 Max = 269.98 Avg =0.46	Written (Mib/s) =58.9

ii. RNDRW

I run combined random read write fileio mode for 5 iterations using a shell script.



Iteration 1

The table below are the latency scores after the 5 iterations:

Iterations	Latency	Throughput
1	Min = 0.01 Max = 475.55 Avg = 0.67	Read (Mib/s) = 23.58 Written (Mib/s) = 15.72
2	Min = 0.01 Max = 296.68 Avg =0.67	Read (Mib/s) =22.45 Written (Mib/s) =16.79
3	Min = 0.01 Max = 204.43 Avg =0.70	Read (Mib/s) =22.6 Written (Mib/s) =15.8
4	Min = 0.01 Max = 593.98 Avg =0.68	Read (Mib/s) =24.33 Written (Mib/s) =12.34
5	Min = 0.01 Max = 150.76 Avg =0.66	Read (Mib/s) =23.66 Written (Mib/s) =11.2

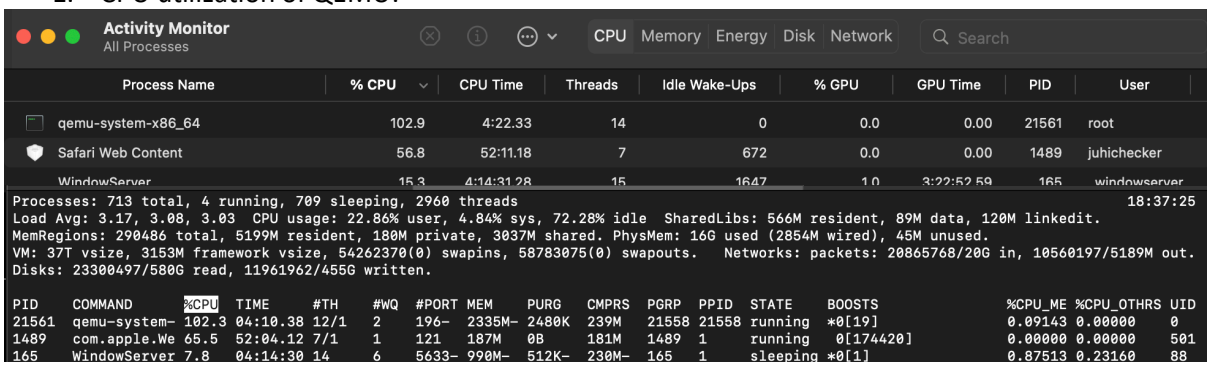
Conclusion:

1. **In CPU mode test it is observed that as the number of max prime value increases the latency of QEMU and Docker also increases. However the latency of QEMU is higher than that of Docker for almost all the cases.**
2. **In File IO mode test it is observed that for sequential write operation latency of QEMU was almost similar to that of Docker. In combined random read write operation the latency for docker was less than that of QEMU.**
3. **The results are similar to first configuration results.**

Analysis of Performance Data

QEMU Analysis:

1. CPU utilization of QEMU:



Percentage of CPU used = 102.9
Kernel Usage

User = 22.86%
System = 4.8%
Idle = 72.28%

2. Disk Utilization
    a. 3Gb memory and 1 core

i. Sequential write
    1. Read (MiB/s) = 0.0
    2. Written (MiB/s) = 34.654
ii. Combined random read write
    1. Read (MiB/s) = 16.79
    2. Written (MiB/s) = 11.196

b. 6Gb memory and 2 core
    i. Sequential write
        1. Read (MiB/s) = 0.0
        2. Written (MiB/s) = 31.14

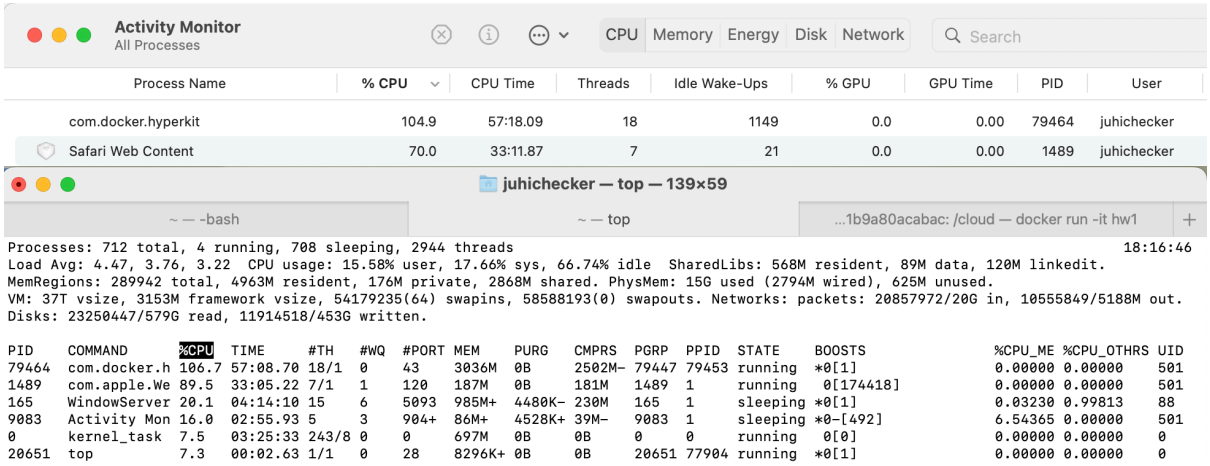
    ii. Combined random read write
        1. Read (MiB/s) = 18.95
        2. Written (MiB/s) = 12.638

c. 9Gb memory and 4 core
    i. Sequential write
        1. Read (MiB/s) = 0.0
        2. Written (MiB/s) = 29.008

    ii. Combined random read write
        1. Read (MiB/s) = 17.706
        2. Written (MiB/s) = 11.872

<u>Docker Analysis:</u>

1. CPU utilization of Docker:



Percentage of CPU used = 104.9
Kernel Usage
    User = 15.58%
    System = 17.66%
    Idle = 66.74%

2. Disk Utilization
    a. 3Gb memory and 1 core
        i. Sequential write
            1. Read (MiB/s) = 0.0
            2. Written (MiB/s) = 39.05

ii. Combined random read write
    1. Read (MiB/s) = 21.07
    2. Written (MiB/s) = 14.04

b. 6Gb memory and 2 core
    i. Sequential write
        1. Read (MiB/s) = 0.0
        2. Written (MiB/s) = 53.96

    ii. Combined random read write
        1. Read (MiB/s) = 25.07
        2. Written (MiB/s) = 15.60

c. 9Gb memory and 4 core
    i. Sequential write
        1. Read (MiB/s) = 0.0
        2. Written (MiB/s) = 53.72

    ii. Combined random read write
        1. Read (MiB/s) = 23.96
        2. Written (MiB/s) = 15.97

Conclusion for CPU Utilization:

1. **It was observed that in QEMU utilised the cpu a bit less than Docker. Moreover, with respect to kernel usage QEMU used more of user space and being idle when compared to Docker whereas, system was utilized more by Docker.**

Conclusion for Disk Utilization:

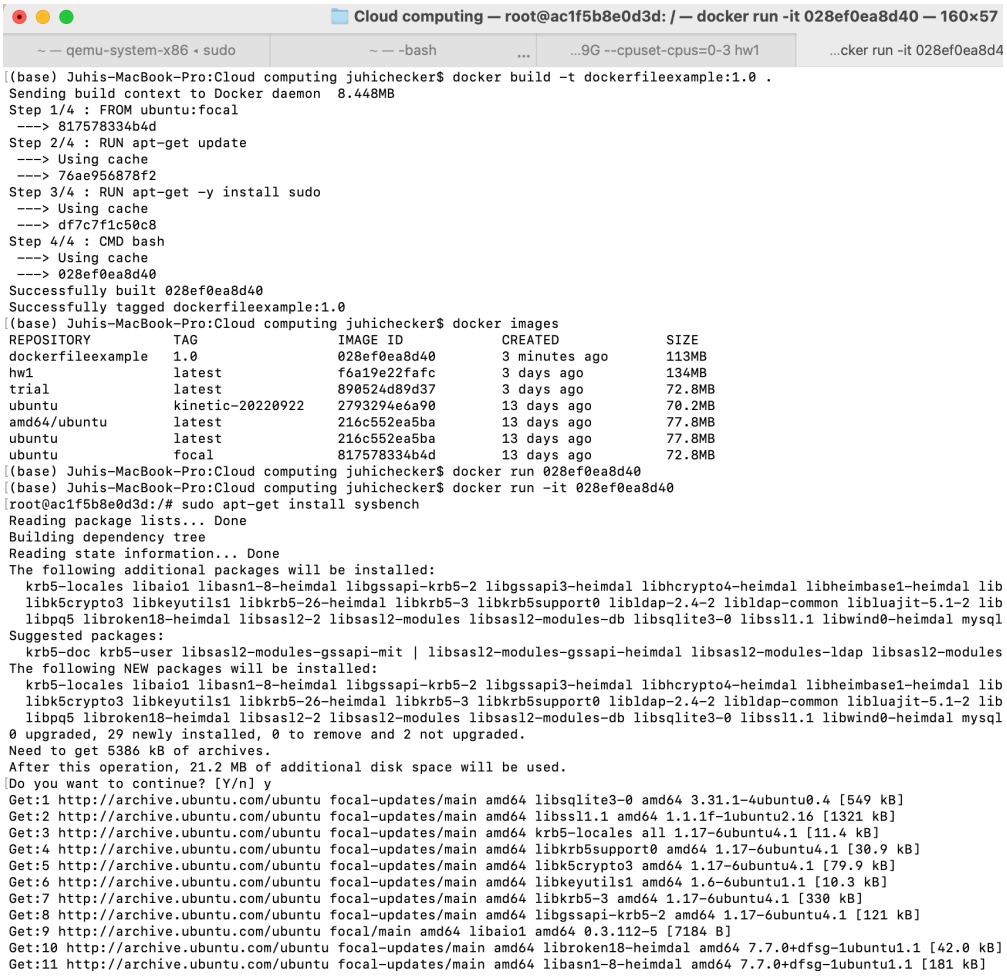
1. **It was observed that in QEMU sequential write, write operation's values decreased as the memory size increased along with cores. Further, combined random read write operation values had no significant changes on increasing the memory and core.**
2. **It was observed that in Docker sequential write, write operation's values increased little and remained constant later as the memory size along with cores. Further, combined random read write operation values had no significant changes on increasing the memory and core like QEMU.**
3. **Between QEMU and Docker, QEMU performed better than Docker file io operations making QEMU faster than Docker.**

Docker File Implementation

I have created a docker image using following commands:

1. vim Dockerfile
2. Add the commands to be run. I have used From, Run and CMD. Please find the file in github
3. docker build -t <containerName>:<Tag> .
4. docker images
5. docker run <ContainerID>

6. Check if sudo was installed
7. The output is attached below.



Shell Scripts:

Account – checkerjuhi21
Repository Name – CloudComputing
Folder Name – HW1
GitHub link – https://github.com/checkerjuhi21/CloudComputing.git
Commit ID - `f8396eb0d00cc06c571537cec2f4f43740a3f838`

Commit id may change after I push this pdf into the github. Please refer to commit id in the text entry that I submit it on canvas.