Amarjot Singh Bhullar
CSEN 241 : Cloud Computing
Homework – 1

Different disk images for QEMU –

1. Raw image
    a. Command - qemu-img create -f raw disk_raw.img 20G
2. Qcow2 image
    a. Command - qemu-img create -f qcow2 disk_qcow2.img 20G
3. Encrypted qcow2 image
    a. Command - qemu-img create --object secret,id=sec0,data=abc123 -f qcow2 -o encrypt.format=luks,encrypt.key-secret=sec0 disk_qcow_encrypted.qcow2 20G

Proof of creation of disks for QEMU –

1. Both Qcow2 and raw image -

```
amarjotsinghbhullar@Amarjots-MacBook-Air Homework 1 % qemu-img create -f raw disk_raw.img 20G

Formatting 'disk_raw.img', fmt=raw size=21474836480
amarjotsinghbhullar@Amarjots-MacBook-Air Homework 1 % qemu-img create -f qcow2 disk_qcow2.img 20G

Formatting 'disk_qcow2.img', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib size=21474836480 lazy_refcounts=off refcount_bits=16
amarjotsinghbhullar@Amarjots-MacBook-Air Homework 1 %
```

2. Encrypted qcow2 image –

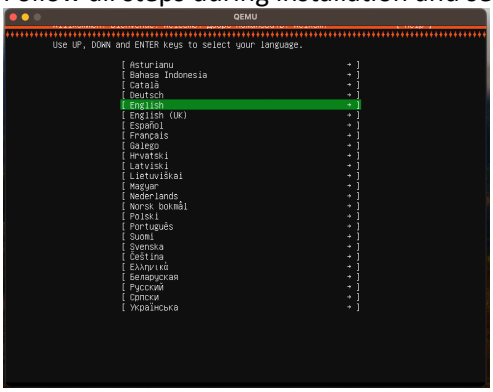
```
amarjotsinghbhullar@Amarjots-MacBook-Air ~ % qemu-img create --object secret,id=sec0,data=abc123 -f qc
ow2 -o encrypt.format=luks,encrypt.key-secret=sec0 disk_qcow_encrypted.qcow2 20G

Formatting 'disk_qcow_encrypted.qcow2', fmt=qcow2 encrypt.format=luks encrypt.key-secret=sec0 cluster_
size=65536 extended_l2=off compression_type=zlib size=21474836480 lazy_refcounts=off refcount_bits=16
amarjotsinghbhullar@Amarjots-MacBook-Air ~ %
```

QEMU VM installation and required setup steps

1. Install QEMU for macbook using command – **brew install qemu**
2. Create all required disks using the command given above.
3. Install ubuntu server in qcow2 disk using command -

```
amarjotsinghbhullar@Amarjots-MacBook-Air ~ % qemu-system-aarch64 \
-accel hvf -cpu cortex-a57 -M virt,highmem=off -m 2048 -smp 2 \
-drive file=/opt/homebrew/Cellar/qemu/8.2.1/share/qemu/edk2-aarch64-code.fd,if=pflash,format=raw,readonly=on \
-drive if=none,file=disk_qcow2.img,format=qcow2,id=hd0 \
-device virtio-blk-device,drive=hd0,serial="dummyserial" \
-device virtio-net-device,netdev=net0 \
-netdev user,id=net0 \
-vga none -device ramfb \
-cdrom ubuntu-20.04.5-live-server-arm64.iso \
-device usb-ehci -device usb-kbd -device usb-mouse -usb
```

4. Follow all steps during installation and select required options such as language -



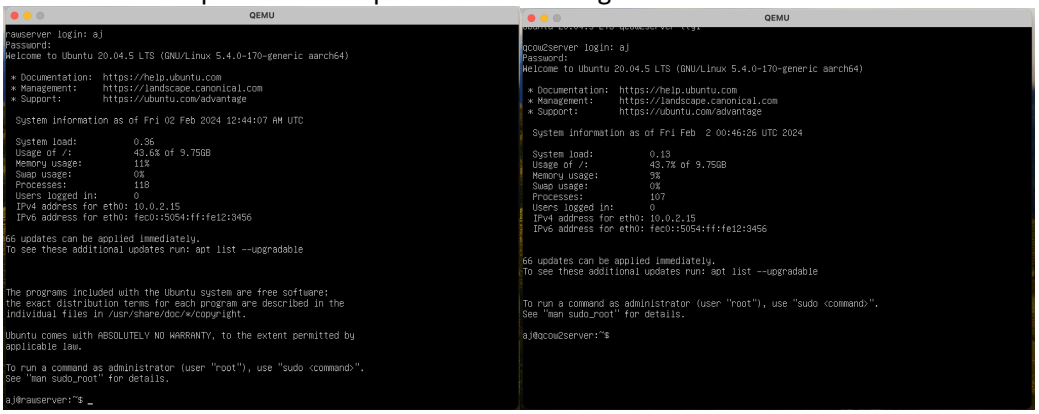
5. Install ubuntu server in raw disk using command –

```
amarjotsinghbhullar@Amarjots-MacBook-Air ~ % qemu-system-aarch64 \
-accel hvf -cpu cortex-a57 -M virt,highmem=off -m 2048 -smp 2 \
-drive file=/opt/homebrew/Cellar/qemu/8.2.1/share/qemu/edk2-aarch64-code.fd,if=pflash,format=raw,readonly=on \
-drive if=none,file=disk_raw.img,format=raw,id=hd0 \
-device virtio-blk-device,drive=hd0,serial="dummyserial" \
-device virtio-net-device,netdev=net0 \
-netdev user,id=net0 \
-vga none -device ramfb \
-cdrom ubuntu-20.04.5-live-server-arm64.iso \
-device usb-ehci -device usb-kbd -device usb-mouse -usb
```

6. Reboot into installed ubuntu in qcow2 disk using command. Change for qcow2 to raw to boot into raw disk –

```
amarjotsinghbhullar@Amarjots-MacBook-Air ~ % qemu-system-aarch64 \
-accel hvf -cpu cortex-a57 -M virt,highmem=off -m 2048 -smp 2 \
-drive file=/opt/homebrew/Cellar/qemu/8.2.1/share/qemu/edk2-aarch64-code.fd,if=pflash,format=raw,readonly=on \
-drive if=none,file=disk_qcow2.img,format=qcow2,id=hd0 \
-device virtio-blk-device,drive=hd0,serial="dummyserial" \
-device virtio-net-device,netdev=net0 \
-netdev user,id=net0 \
-vga none -device ramfb \
-device usb-ehci -device usb-kbd -device usb-mouse -usb
```

7. Keep starting the server with different CPU cores and ram options. For my testing purposes I have used these four combinations of CPU and ram –
    a. -m 2048 -smp 2
    b. -m 1024 -smp 2
    c. -m 2048 -smp 4
    d. -m 1024 -smp 4
8. Post installation proof in both qcow2 and raw image -



9. Install sysbench in all ubuntu server installation using commands –
    a. sudo apt update

b. sudo apt install sysbench

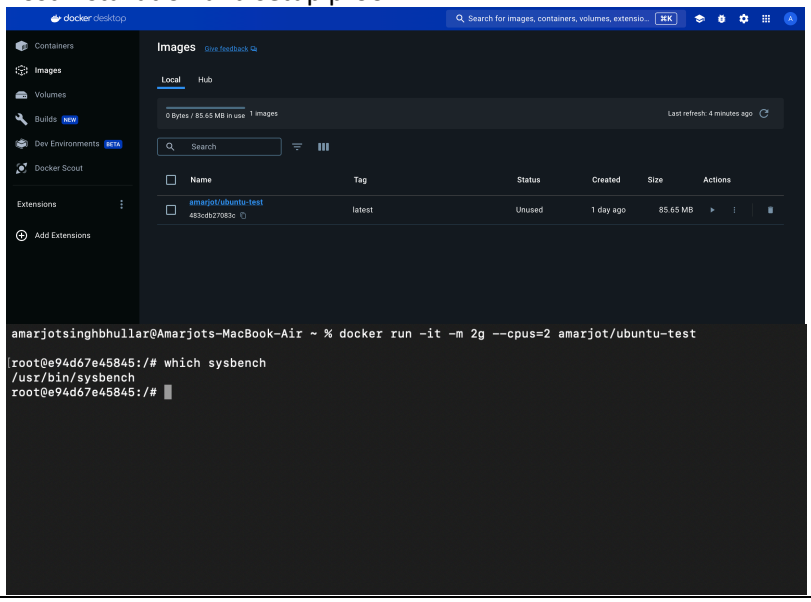
Docker Installation and required setup steps

1. **Docker installation steps** –
   a. Visit https://docs.docker.com/desktop/install/mac-install/ and download docker desktop using Docker Desktop for Mac with Apple silicon option.
   b. Click on downloaded dmg file and move it to applications.
2. Create a new Dockerfile with the following code to use ubuntu image and install sysbench in it. -

```
# Use Ubuntu 20.04 LTS as base image

FROM ubuntu:20.04



# Avoid prompts from apt

ARG DEBIAN_FRONTEND=noninteractive



# Update and install sysbench

RUN apt-get update && \

    apt-get install -y sysbench && \

    # Clean up to reduce image size

    apt-get clean && \

    rm -rf /var/lib/apt/lists/*



# Specify the default command for the container

CMD ["bash"]
```

3. Build the ubuntu image with sysbench server using command - **docker build -t amarjot/ubuntu-test:latest .**
4. Login into docker using command – **docker login**
5. Push the image into docker hub using command - **docker push amarjot/ubuntu-test**

6. Run the image in docker using command - **docker run -it -m 2g --cpus=2 amarjot/ubuntu-test**
7. Keep modifying the parameters for docker using -m and -cpus options. For my tests I have used the following options –
    a. -m 2g -cpus=2
    b. -m 1g -cpus=2
    c. -m 2g -cpus=4
    d. -m 1g -cpus=4
8. Post installation and setup proof –

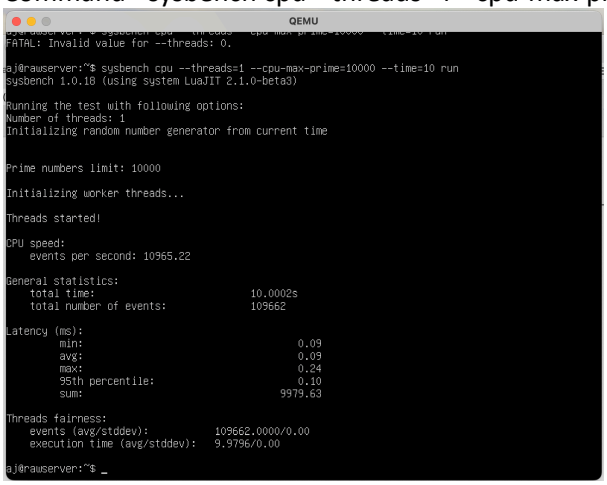


Proof Of Experiments using Sysbench

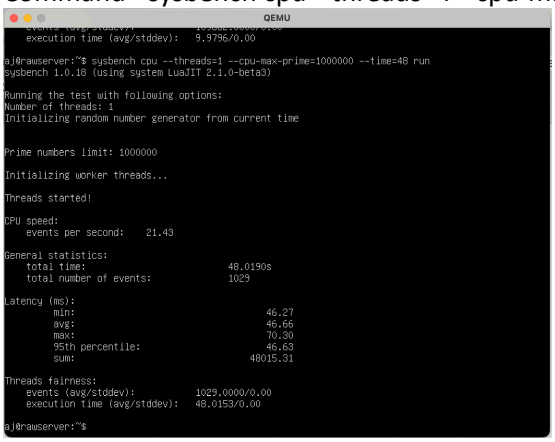
1. QEMU VM using RAW disk image (CPU – 2 cores RAM – 2Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run

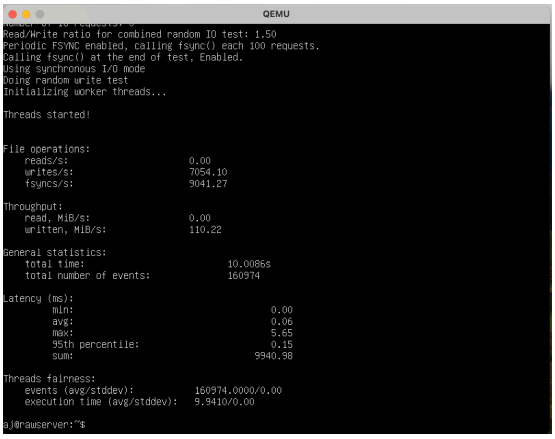


Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run

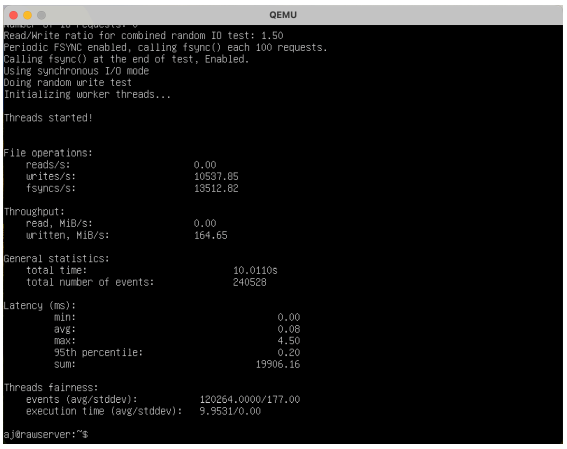


b. FILE I/O-

Prepare command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup

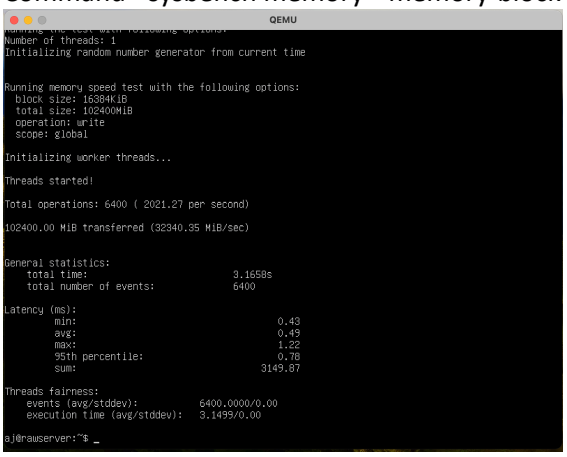


Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2--file-total-size=4G --file-test-mode=rndwr cleanup

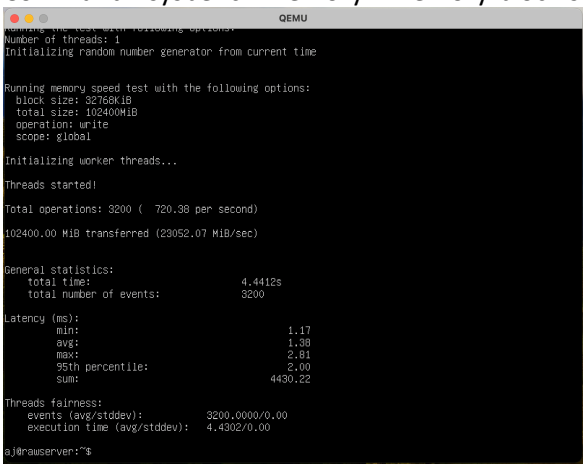


c. Memory test –

Command - sysbench memory --memory-block-size=16M run



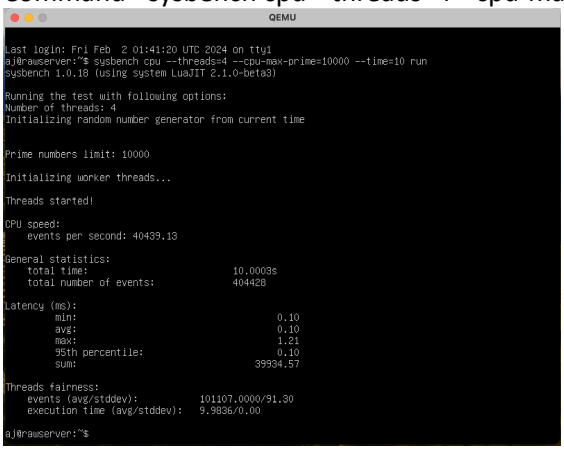
Command - sysbench memory --memory-block-size=32M run



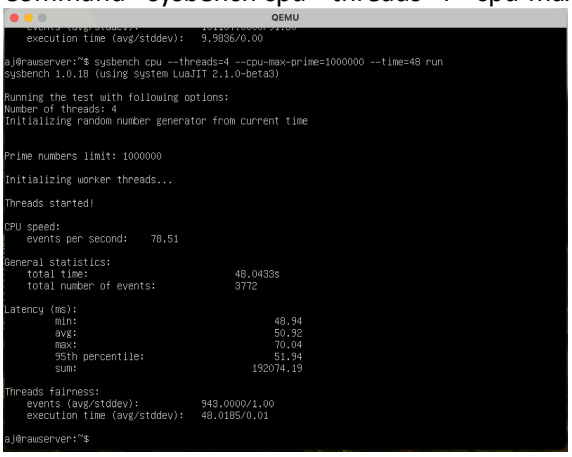
2. QEMU VM using RAW disk image (CPU – 4 cores RAM – 2Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run



Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



b. FILE I/O-

Prepare command –

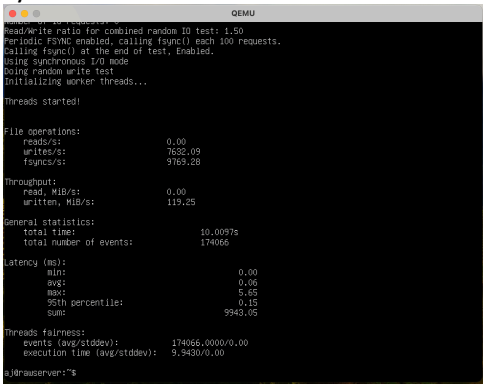
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare

Run command –

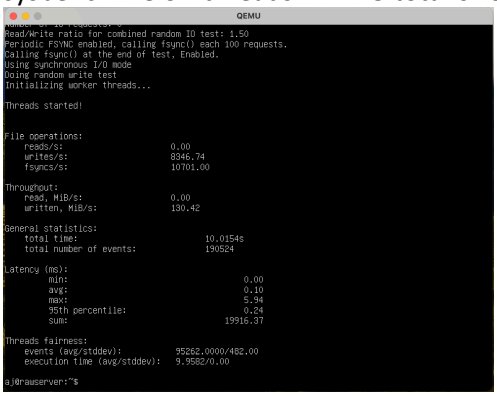
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run

Cleanup Command –

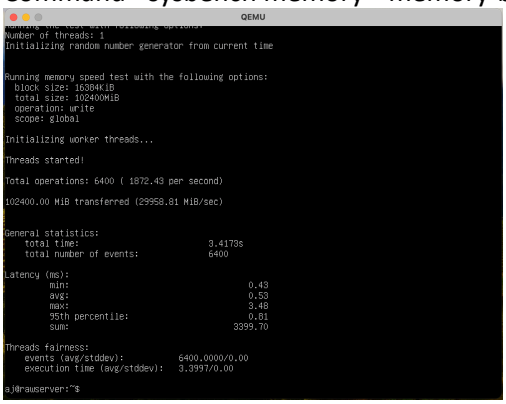
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



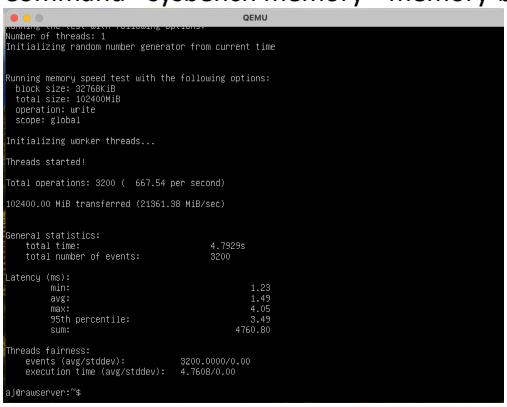
Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –
Command - sysbench memory --memory-block-size=16M run



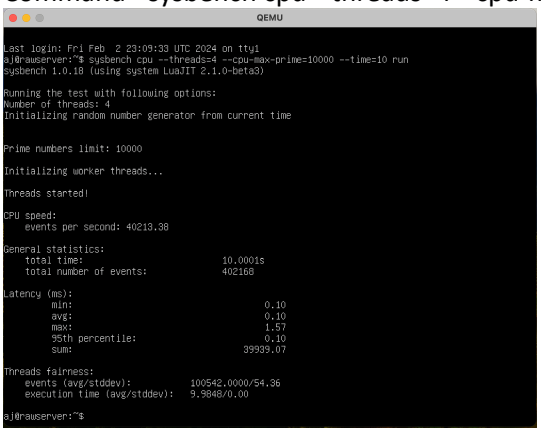
Command - sysbench memory --memory-block-size=32M run



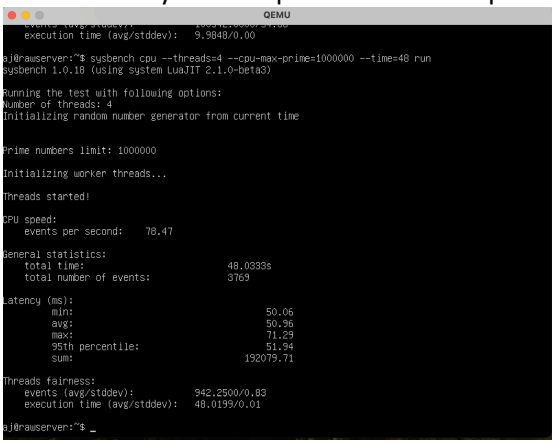
3. QEMU VM using RAW disk image (CPU – 4 cores RAM – 1Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run

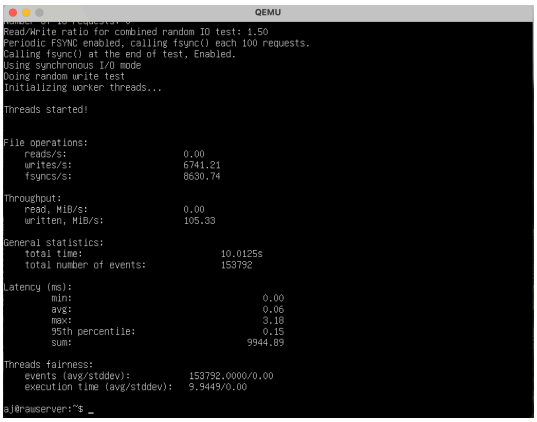


Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run

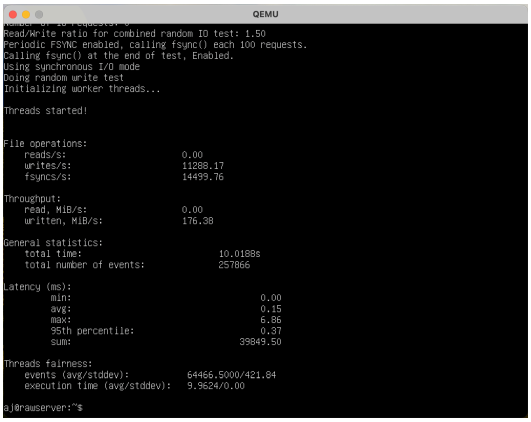


b. FILE I/O-

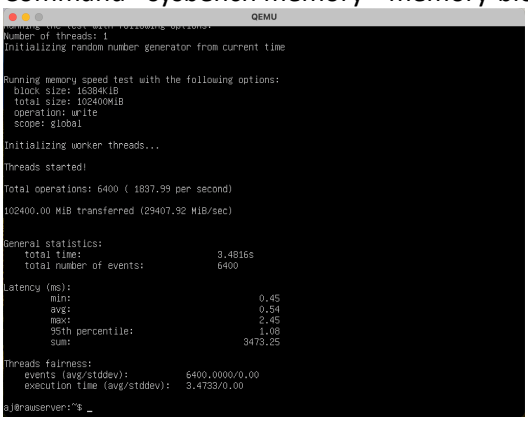
Prepare command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



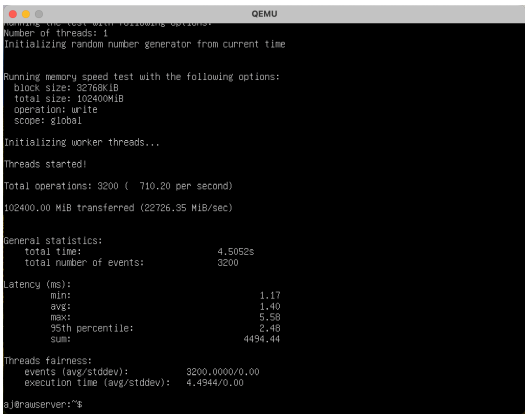
Prepare command –
sysbench fileio --threads=4--file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=4 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=4--file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –
Command - sysbench memory --memory-block-size=16M run



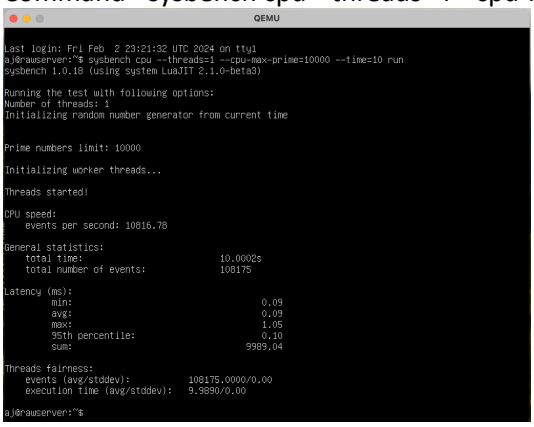
Command - sysbench memory --memory-block-size=32M run



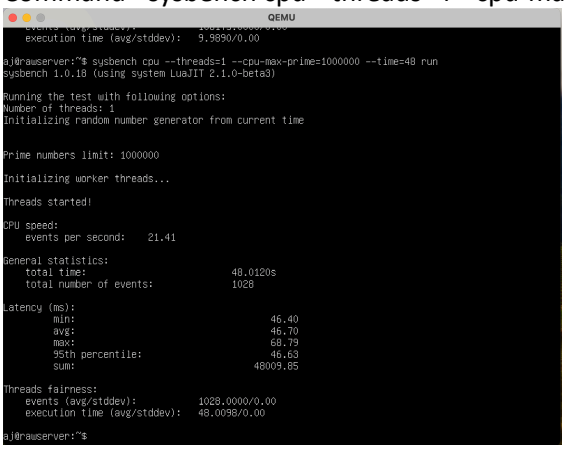
4. QEMU VM using RAW disk image (CPU – 2 cores RAM – 1Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run



Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



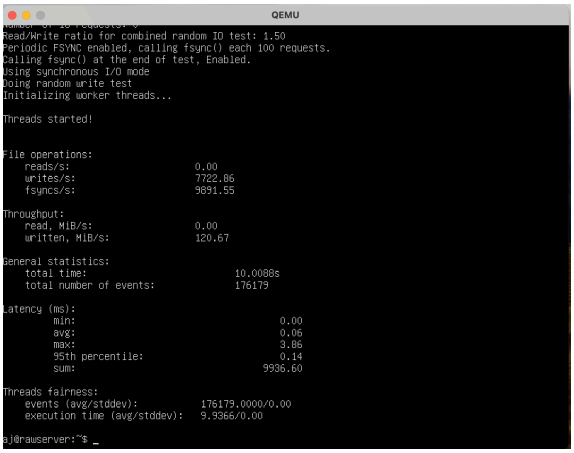
b. FILE I/O-

Prepare command –

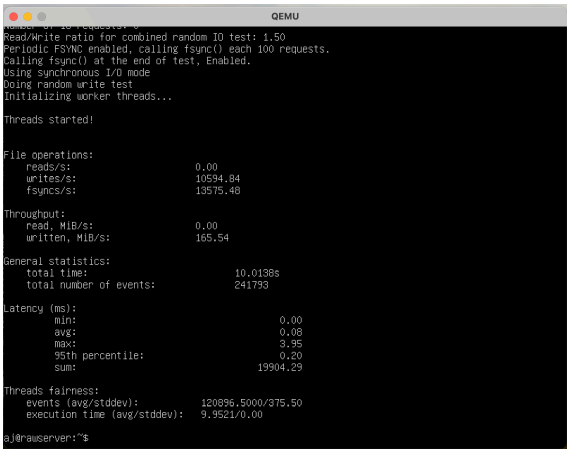
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare

Run command –

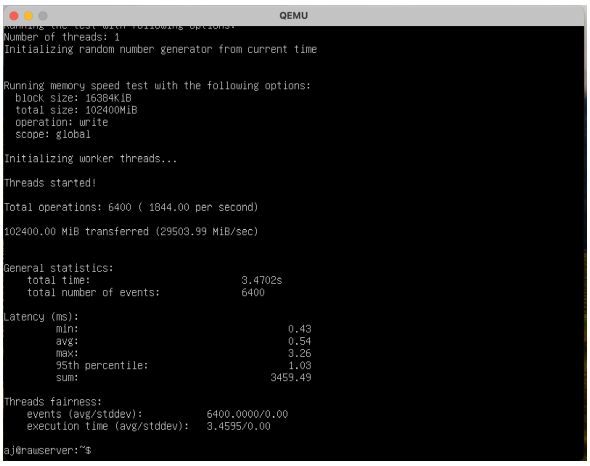
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



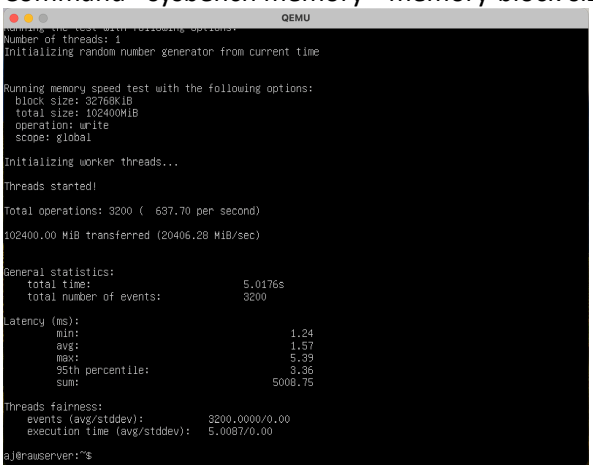
Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –
Command - sysbench memory --memory-block-size=16M run



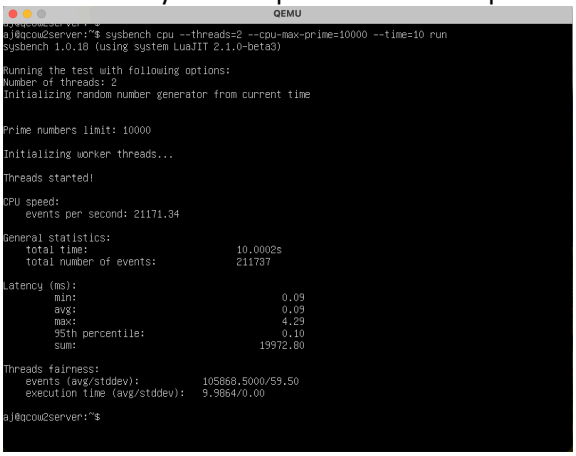
Command - sysbench memory --memory-block-size=32M run



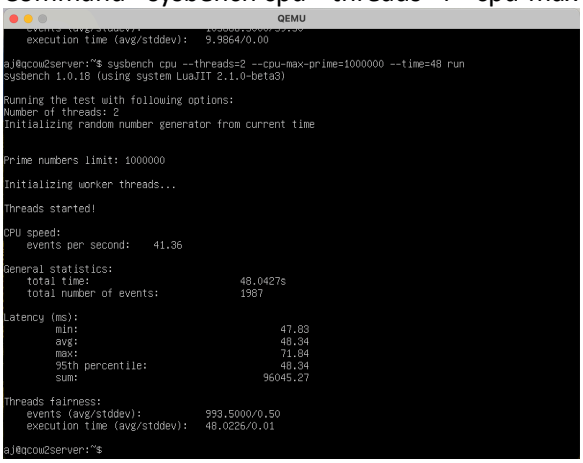
5. QEMU VM using QCOW2 disk image (CPU – 2 cores RAM – 2Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run

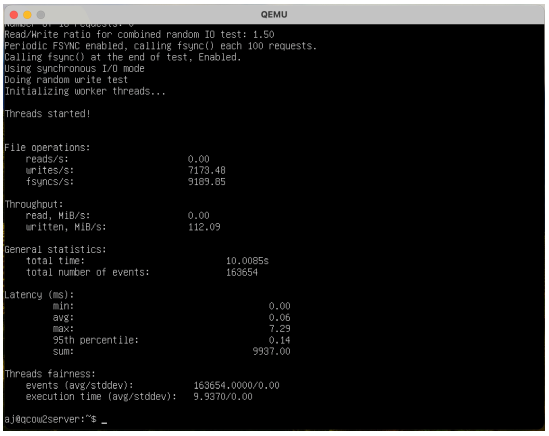


Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run

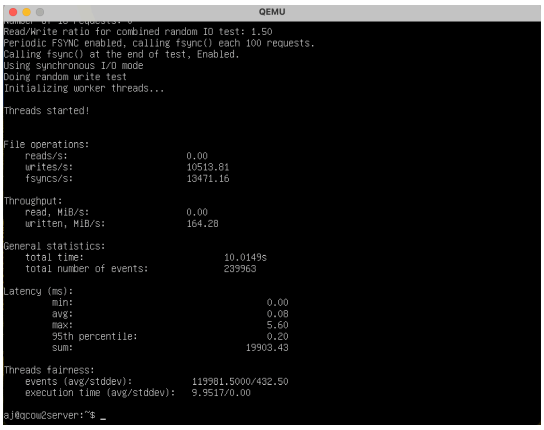


b. FILE I/O-

Prepare command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup

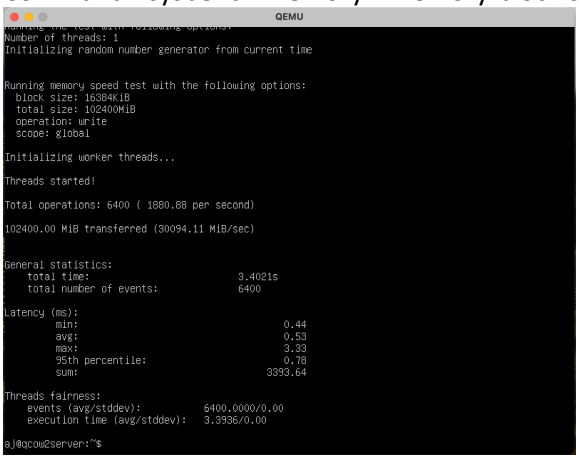


Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2--file-total-size=4G --file-test-mode=rndwr cleanup

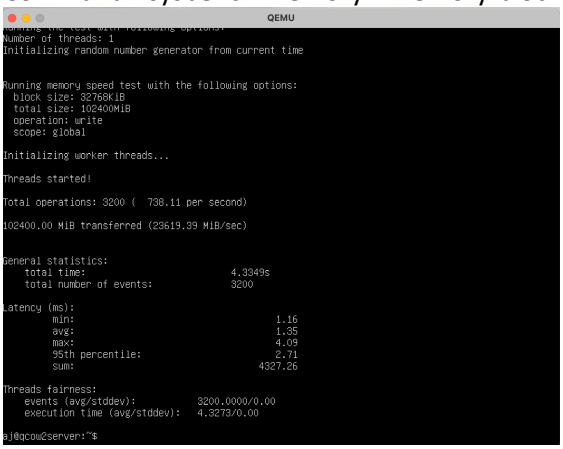


c. Memory test –

Command - sysbench memory --memory-block-size=16M run



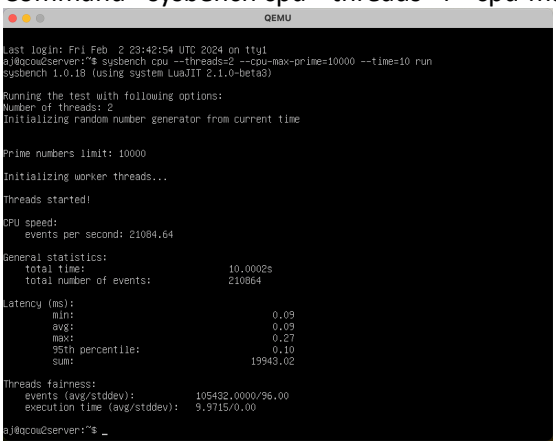
Command - sysbench memory --memory-block-size=32M run



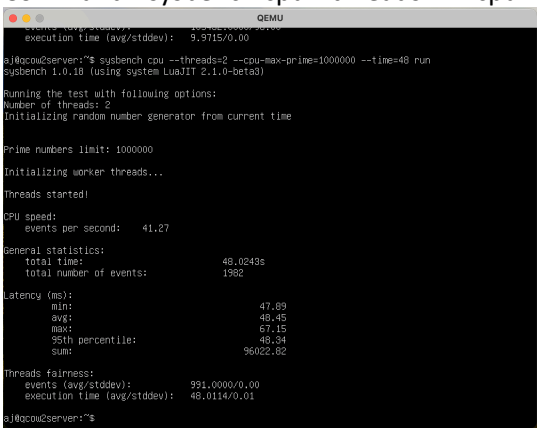
6. QEMU VM using QCOW2 disk image (CPU – 4 cores RAM – 2Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run



Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



b. FILE I/O-

Prepare command –

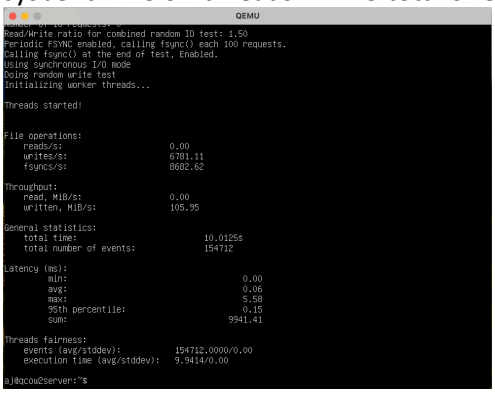
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare

Run command –

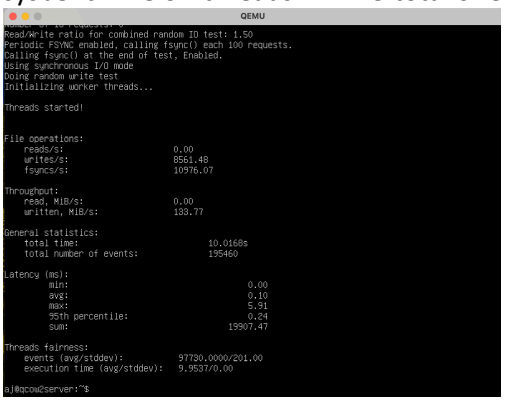
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run

Cleanup Command –

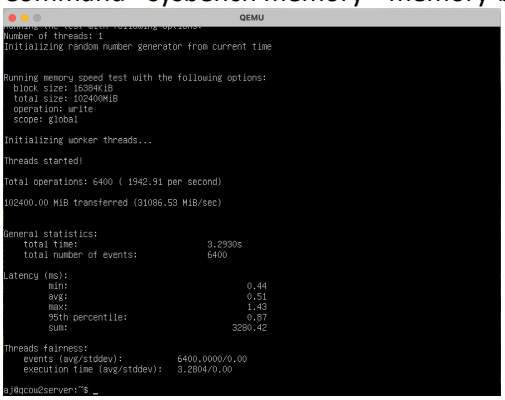
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



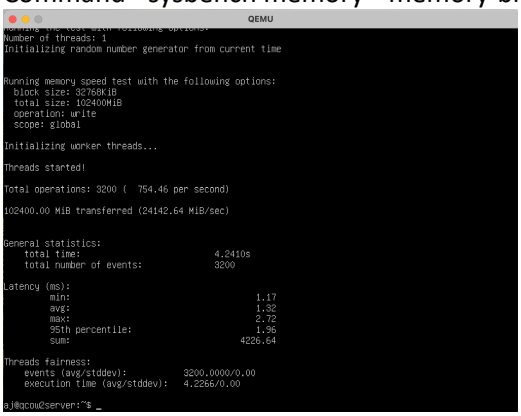
Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –
Command - sysbench memory --memory-block-size=16M run



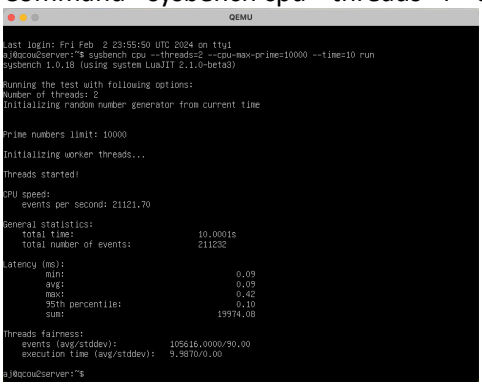
Command - sysbench memory --memory-block-size=32M run



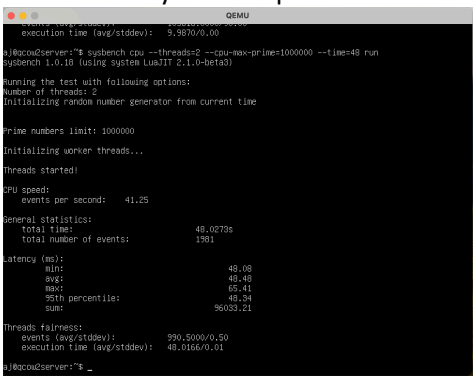
7. QEMU VM using QCOW2 disk image (CPU – 4 cores RAM – 1Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run

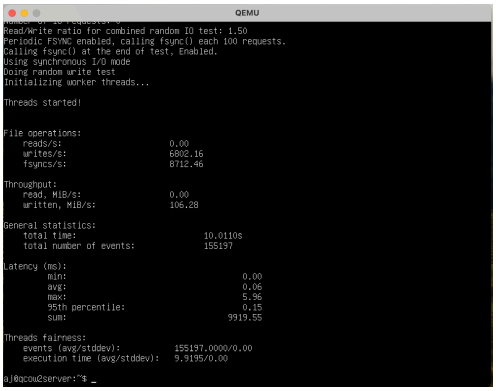


Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run

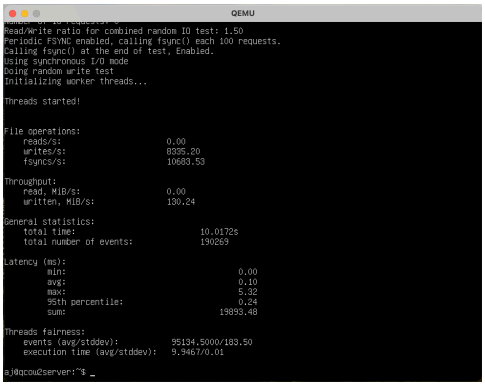


b. FILE I/O-

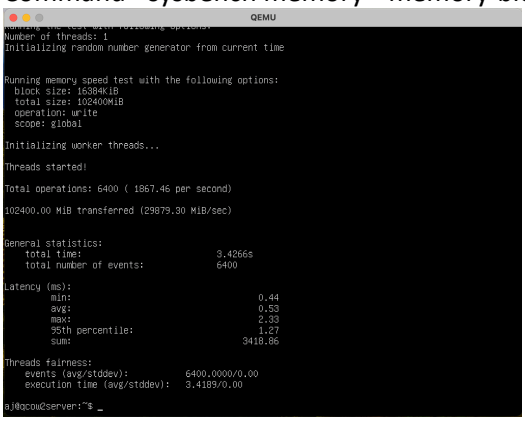
Prepare command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



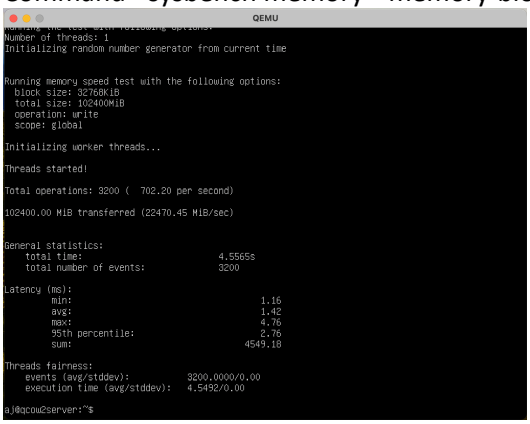
Prepare command –
sysbench fileio --threads=4--file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=4 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=4--file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –
Command - sysbench memory --memory-block-size=16M run



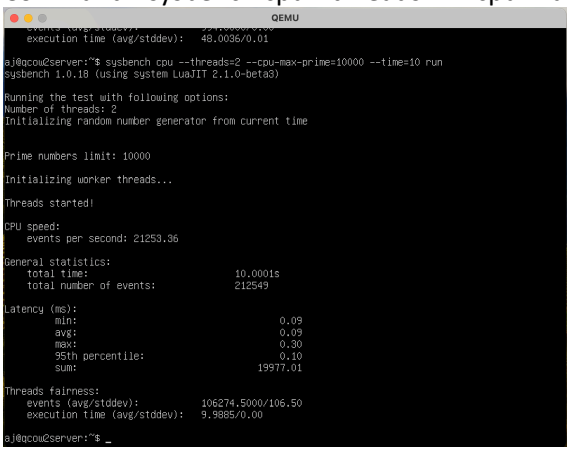
Command - sysbench memory --memory-block-size=32M run



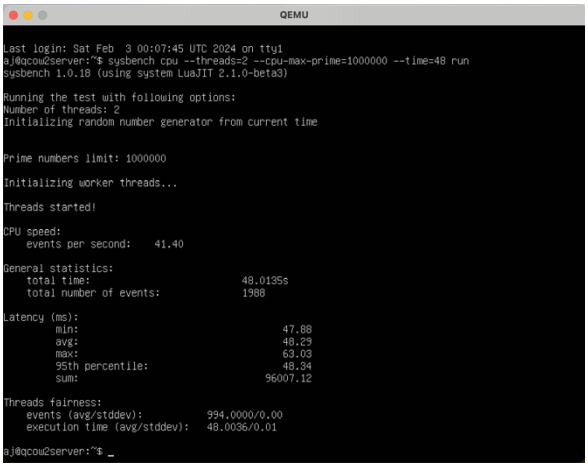
8. QEMU VM using QCOW2 disk image (CPU – 2 cores RAM – 1Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run



Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



b. FILE I/O-

Prepare command –

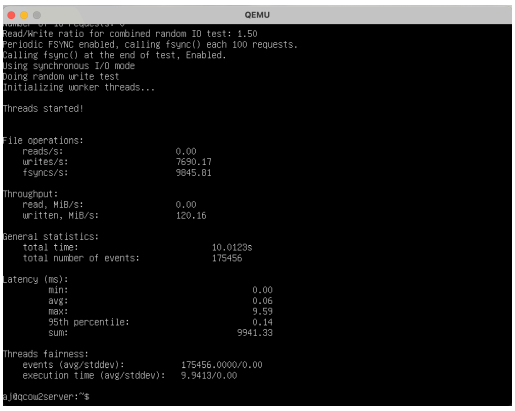
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare

Run command –

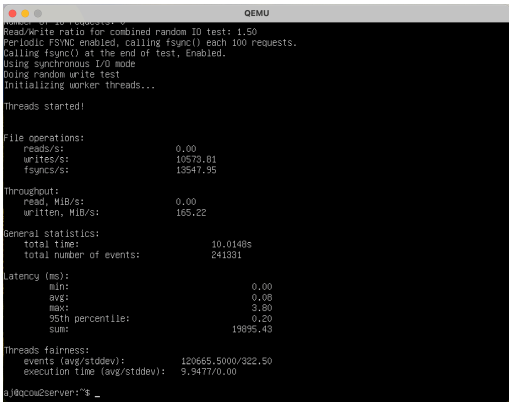
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run

Cleanup Command –

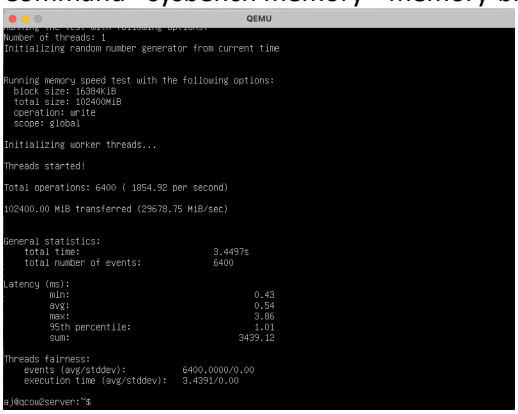
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



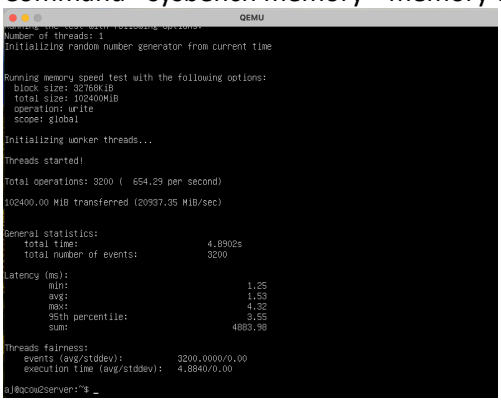
Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –
Command - sysbench memory --memory-block-size=16M run



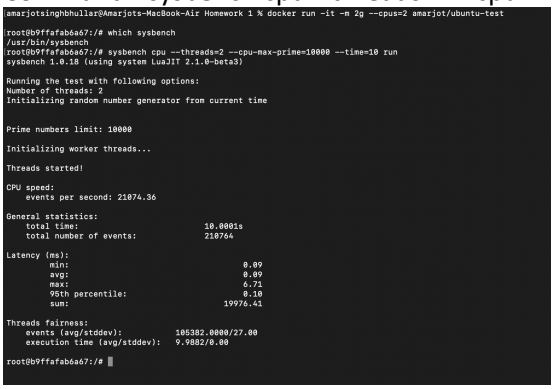
Command - sysbench memory --memory-block-size=32M run



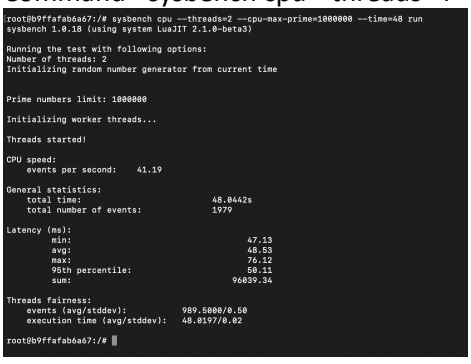
9. Docker (CPU – 2 cores RAM – 2Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run



Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



b. FILE I/O-

Prepare command –

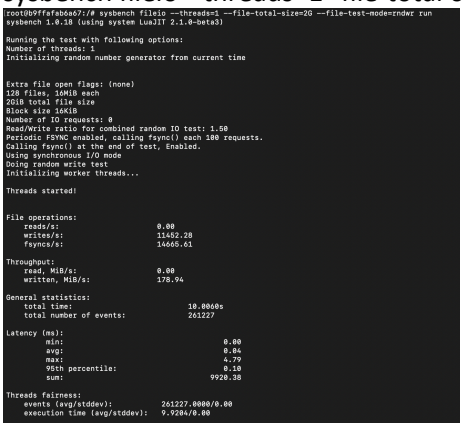
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare

Run command –

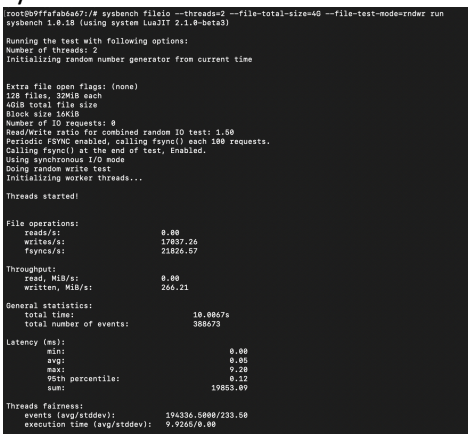
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run

Cleanup Command –

sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2--file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –
Command - sysbench memory --memory-block-size=16M run

```
root@b9ffafab6a67:/# sysbench memory --memory-block-size=16M run
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


Running memory speed test with the following options:
  block size: 16384KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 6400 ( 2074.68 per second)

102400.00 MiB transferred (33194.82 MiB/sec)


General statistics:
    total time:                          3.0837s
    total number of events:              6400

Latency (ms):
         min:                                    0.43
         avg:                                    0.48
         max:                                    1.57
         95th percentile:                        0.54
         sum:                                 3074.68

Threads fairness:
    events (avg/stddev):           6400.0000/0.00
    execution time (avg/stddev):   3.0747/0.00
```

Command - sysbench memory --memory-block-size=32M run

```
root@b9ffafab6a67:/# sysbench memory --memory-block-size=32M run
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


Running memory speed test with the following options:
  block size: 32768KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 3200 (  739.77 per second)

102400.00 MiB transferred (23672.63 MiB/sec)


General statistics:
    total time:                          4.3246s
    total number of events:              3200

Latency (ms):
         min:                                    1.21
         avg:                                    1.35
         max:                                    2.87
         95th percentile:                        1.39
         sum:                                 4318.22

Threads fairness:
    events (avg/stddev):           3200.0000/0.00
    execution time (avg/stddev):   4.3182/0.00
```

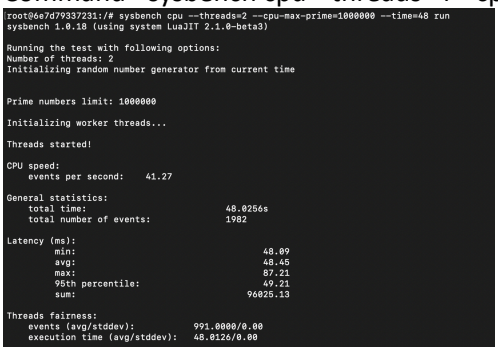
10. Docker (CPU – 4 cores RAM – 2Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run

```
root@6e7d79337231:/# sysbench cpu --threads=2 --cpu-max-prime=10000 --time=10 run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time


Prime numbers limit: 10000

Initializing worker threads...

Threads started!

CPU speed:
    events per second: 21155.74

General statistics:
    total time:                          10.0002s
    total number of events:              211580

Latency (ms):
         min:                                    0.09
         avg:                                    0.09
         max:                                    0.61
         95th percentile:                        0.10
         sum:                                19976.36

Threads fairness:
    events (avg/stddev):           105790.0000/74.00
    execution time (avg/stddev):   9.9882/0.00
```

Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



b. FILE I/O-

Prepare command –

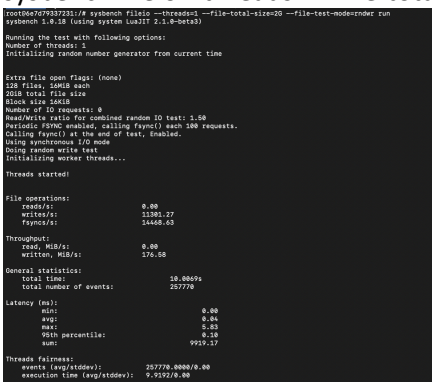
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare

Run command –

sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run

Cleanup Command –

sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



Prepare command –

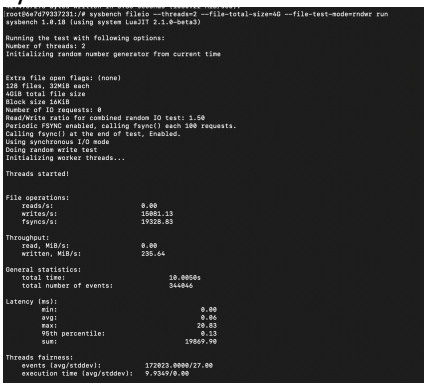
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare

Run command –

sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run

Cleanup Command –

sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr cleanup



c. Memory test –

Command - sysbench memory --memory-block-size=16M run

```
root@6e7d79337231:/# sysbench memory --memory-block-size=16M run
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


Running memory speed test with the following options:
  block size: 16384KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 6400 ( 2246.33 per second)

102400.00 MiB transferred (35941.33 MiB/sec)


General statistics:
    total time:                          2.8479s
    total number of events:              6400

Latency (ms):
         min:                                    0.42
         avg:                                    0.44
         max:                                    1.76
         95th percentile:                        0.46
         sum:                                 2842.00

Threads fairness:
    events (avg/stddev):           6400.0000/0.00
    execution time (avg/stddev):   2.8420/0.00
```

Command - sysbench memory --memory-block-size=32M run

```
root@6e7d79337231:/# sysbench memory --memory-block-size=32M run
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


Running memory speed test with the following options:
  block size: 32768KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 3200 (  843.89 per second)

102400.00 MiB transferred (27004.53 MiB/sec)


General statistics:
    total time:                          3.7911s
    total number of events:              3200

Latency (ms):
         min:                                    1.12
         avg:                                    1.18
         max:                                    3.53
         95th percentile:                        1.27
         sum:                                 3784.97

Threads fairness:
    events (avg/stddev):           3200.0000/0.00
    execution time (avg/stddev):   3.7850/0.00
```

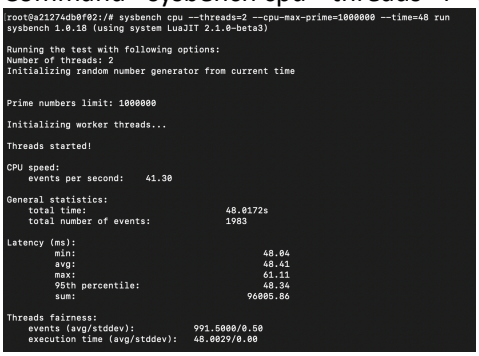
11. Docker (CPU – 4 cores RAM – 1Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run

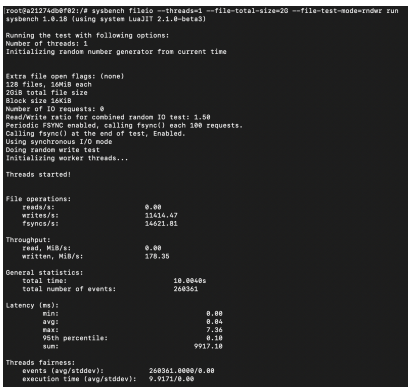
```
root@a21274db0f02:/# sysbench cpu --threads=2 --cpu-max-prime=10000 --time=10 run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time


Prime numbers limit: 10000

Initializing worker threads...

Threads started!

CPU speed:
    events per second: 21165.88

General statistics:
    total time:                          10.0001s
    total number of events:              211680

Latency (ms):
         min:                                    0.09
         avg:                                    0.09
         max:                                    1.11
         95th percentile:                        0.10
         sum:                                19979.40

Threads fairness:
    events (avg/stddev):           105840.0000/11.00
    execution time (avg/stddev):   9.9897/0.00
```

Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



b. FILE I/O-

Prepare command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup



Prepare command –
sysbench fileio --threads=4--file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=4 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=4--file-total-size=4G --file-test-mode=rndwr cleanup

```
root@a21274db0f02:/# sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time


Extra file open flags: (none)
128 files, 32MiB each
4GiB total file size
Block size 16KiB
Number of IO requests: 0
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 write test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      0.00
    writes/s:                     16644.37
    fsyncs/s:                     21319.78

Throughput:
    read, MiB/s:                  0.00
    written, MiB/s:               260.07

General statistics:
    total time:                          10.0086s
    total number of events:              377742

Latency (ms):
         min:                                    0.00
         avg:                                    0.05
         max:                                    7.04
         95th percentile:                        0.13
         sum:                                19861.07

Threads fairness:
    events (avg/stddev):           189871.0000/569.80
    execution time (avg/stddev):   9.9259/0.00
```

c. Memory test –

Command - sysbench memory --memory-block-size=16M run

```
root@a21274db0f02:/# sysbench memory --memory-block-size=16M run
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


Running memory speed test with the following options:
  block size: 16384KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 6400 ( 2240.13 per second)

102400.00 MiB transferred (35842.13 MiB/sec)


General statistics:
    total time:                          2.8560s
    total number of events:              6400

Latency (ms):
         min:                                    0.42
         avg:                                    0.45
         max:                                    1.85
         95th percentile:                        0.47
         sum:                                 2849.88

Threads fairness:
    events (avg/stddev):           6400.0000/0.00
    execution time (avg/stddev):   2.8499/0.00
```

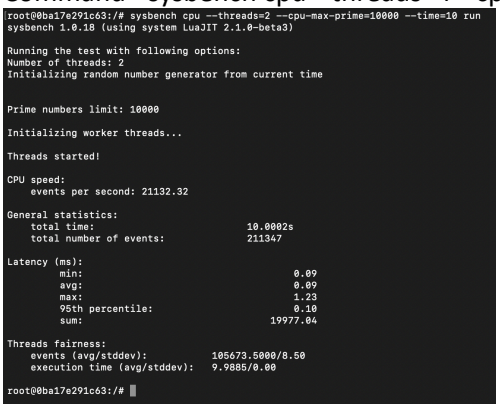
Command - sysbench memory --memory-block-size=32M run

```
root@a21274db0f02:/# sysbench memory --memory-block-size=32M run
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


Running memory speed test with the following options:
  block size: 32768KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 3200 (  849.56 per second)

102400.00 MiB transferred (27185.84 MiB/sec)


General statistics:
    total time:                          3.7658s
    total number of events:              3200

Latency (ms):
         min:                                    1.12
         avg:                                    1.17
         max:                                    2.72
         95th percentile:                        1.21
         sum:                                 3758.74

Threads fairness:
    events (avg/stddev):           3200.0000/0.00
    execution time (avg/stddev):   3.7587/0.00
```

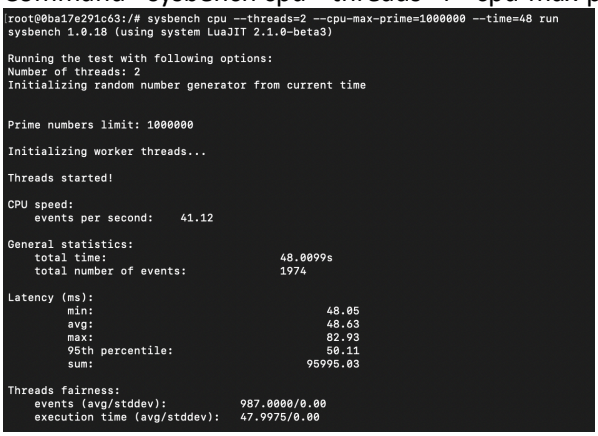
12. Docker (CPU – 2 cores RAM – 1Gb)

a. CPU Test

Command - sysbench cpu --threads=4 --cpu-max-prime=10000 --time=10 run



Command - sysbench cpu --threads=4 --cpu-max-prime=1000000 --time=48 run



b. FILE I/O-

Prepare command –

sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr prepare

Run command –

sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr run

Cleanup Command –

sysbench fileio --threads=1--file-total-size=2G --file-test-mode=rndwr cleanup

```
root@0ba17e291c63:/# sysbench fileio --threads=1 --file-total-size=2G --file-test-mode=rndwr run
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


Extra file open flags: (none)
128 files, 16MiB each
2GiB total file size
Block size 16KiB
Number of IO requests: 0
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 write test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      0.00
    writes/s:                     11420.92
    fsyncs/s:                     14620.48

Throughput:
    read, MiB/s:                  0.00
    written, MiB/s:               178.45

General statistics:
    total time:                          10.0071s
    total number of events:              260493

Latency (ms):
         min:                                    0.00
         avg:                                    0.04
         max:                                    5.39
         95th percentile:                        0.10
         sum:                                 9915.09

Threads fairness:
    events (avg/stddev):           260493.0000/0.00
    execution time (avg/stddev):   9.9151/0.00
```

Prepare command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr prepare
Run command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
Cleanup Command –
sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr cleanup

```
root@0ba17e291c63:/# sysbench fileio --threads=2 --file-total-size=4G --file-test-mode=rndwr run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time


Extra file open flags: (none)
128 files, 32MiB each
4GiB total file size
Block size 16KiB
Number of IO requests: 0
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 write test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      0.00
    writes/s:                     16679.96
    fsyncs/s:                     21373.74

Throughput:
    read, MiB/s:                  0.00
    written, MiB/s:               260.62

General statistics:
    total time:                          10.0053s
    total number of events:              380510

Latency (ms):
         min:                                    0.00
         avg:                                    0.05
         max:                                    5.71
         95th percentile:                        0.12
         sum:                                19851.73

Threads fairness:
    events (avg/stddev):           190255.0000/245.00
    execution time (avg/stddev):   9.9259/0.00
```

c. Memory test –
Command - sysbench memory --memory-block-size=16M run

```
root@0ba17e291c63:/# sysbench memory --memory-block-size=16M run
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


Running memory speed test with the following options:
  block size: 16384KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 6400 ( 2241.93 per second)

102400.00 MiB transferred (35870.88 MiB/sec)


General statistics:
    total time:                          2.8536s
    total number of events:              6400

Latency (ms):
         min:                                    0.42
         avg:                                    0.44
         max:                                    2.47
         95th percentile:                        0.47
         sum:                                 2847.41

Threads fairness:
    events (avg/stddev):           6400.0000/0.00
    execution time (avg/stddev):   2.8474/0.00
```

Command - sysbench memory --memory-block-size=32M run

```
root@0ba17e291c63:/# sysbench memory --memory-block-size=32M run
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


Running memory speed test with the following options:
  block size: 32768KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 3200 (  854.77 per second)

102400.00 MiB transferred (27352.60 MiB/sec)


General statistics:
    total time:                          3.7427s
    total number of events:              3200

Latency (ms):
         min:                                    1.12
         avg:                                    1.17
         max:                                    2.68
         95th percentile:                        1.21
         sum:                                 3736.50

Threads fairness:
    events (avg/stddev):           3200.0000/0.00
    execution time (avg/stddev):   3.7365/0.00
```

Measurements and Analysis of Results

CPU Test:
The following factors are considered when running the CPU test:
a. **--cpu-max-prime**: This setting determines the highest prime number that the CPU tests will look for.
b. **--time**: This parameter determines the test's overall duration in seconds.
c. **--threads**: This setting determines how many parallel threads will be used during the test in order to determine the average, min, max, and standard values of the results, the experiment involves changing the values of the above mentioned parameters and **executing each test case five times**.

FileIO Test:
Following parameters are considered to perform FileIO Test:
a. **--file-total-size**: This option specifies the test file's overall size in megabytes.
--file-test-mode: This option determines what kind of file I/O test will be performed.
--threads: This parameter determines how many threads will be used overall for the process.
In order to determine the average, min, max, and standard values of the results, the experiment involves changing the values of the aforementioned parameters and **executing each test case five times**.

Memory Test:
a. The memory test involves changing the value of the **--memory-block-size parameter** to different sizes and running the test case multiple times for each block size. The purpose is to evaluate memory performance under varying block size conditions.
The primary goal of this memory test is to understand how memory block sizes impact memory subsystem performance. By varying the block size and conducting multiple test runs, you can assess how the system's memory operations, such as reads and writes, behave with different block sizes.
This information helps in optimizing memory usage for specific applications, as different applications may perform better with different memory block sizes.
In order to determine the average, min, max, and standard values of the results, the experiment involves changing the values of the aforementioned parameters and **executing each test case five times**.

1. **For 2gb ram and 2 core processor tests** -

a. CPU Max Prime – 10000

1. **QEMU with raw disk**: The CPU executed an average of 10955 events per second.
2. **QEMU with QCOW2 disk**: Performance improved to an average of 21171 events per second.
3. **Docker with Ubuntu image**: The CPU performance is similar to QCOW2 with an average of 21074 events per second.

The performance increase from the raw disk to QCOW2 and Docker scenarios could be due to differences in disk I/O efficiency, as QCOW2 and Docker are designed to be more efficient with such operations

b. CPU Max Prime - 1000000

1. **QEMU with raw disk**:
    - Events per second: 21.43
    - Total number of events: 1029
2. **QEMU with QCOW2 disk**:
    - Events per second: 41.36
    - Total number of events: 1987
3. **Docker**:
    - Events per second: 41.19
    - Total number of events: 1979

From these results, QEMU with QCOW2 disk and Docker perform similarly, showing a significant increase in events per second compared to QEMU with a raw disk. This suggests that QCOW2 disk format and Docker's storage mechanisms are more efficient for the CPU-intensive tasks in this benchmark.

c. File I/O with file size 2Gb

1. **QEMU with raw disk**:
    - Write operations: 7045.10 per second.
    - Throughput: 110.22 MiB/s
2. **QEMU with QCOW2 disk**:
    - Write operations: 7173.48 per second
    - Throughput: 112.09 MiB/s
3. **Docker**:
    - Write operations: 11462.28 per second.
    - Throughput: 178.94 MiB/s

From the data, Docker provides the highest write operations per second and throughput, indicating better performance in file I/O operations compared to both QEMU raw and QCOW2 disk images.

d. File I/O with file size 4Gb

1. **QEMU with raw disk**:
    - Write operations: 10537.85 per second
    - Throughput: 164.65 MiB/s
2. **QEMU with QCOW2 disk**:
    - Write operations: 10513.81 per second
    - Throughput: 164.28 MiB/s
3. **Docker**:
    - Write operations: 17837.26 per second
    - Throughput: 266.21 MiB/s

Docker shows a significant increase in write operations per second and throughput when using a larger file size for the test, outperforming both QEMU raw and QCOW2 disk images. This suggests that Docker's storage drivers are more efficient when handling larger files.

e. **Memory test with 16M memory block size**

1. **QEMU with raw disk**:
    - Total operations: 6400 (2021.27 per second)
    - Data transferred: 102400.00 MiB (32340.35 MiB/sec)
2. **QEMU with QCOW2 disk**:
    - Total operations: 6400 (1800.88 per second)
    - Data transferred: 102400.00 MiB (30094.11 MiB/sec)
3. **Docker container**:
    - Total operations: 6400 (2074.68 per second)
    - Data transferred: 102400.00 MiB (33194.82 MiB/sec)

From these results, the Docker container showed the highest memory throughput, followed closely by QEMU with a raw disk, and then QEMU with QCOW2 disk. The higher performance in Docker can be attributed to its more direct access to the host system's resources and possibly more efficient memory management.

f. **Memory test with 32M memory block size**

1. **First Image (QEMU with raw disk)**:
    - Total operations: 3200 (720.38 per second)
    - Data transferred: 102400.00 MiB (23052.07 MiB/sec)
2. **Second Image (QEMU with QCOW2 disk)**:
    - Total operations: 3200 (739.77 per second)
    - Data transferred: 102400.00 MiB (23672.63 MiB/sec)
3. **Third Image (Docker container)**:
    - Total operations: 3200 (736.11 per second)
    - Data transferred: 102400.00 MiB (23619.99 MiB/sec)

In these tests, the QCOW2 disk image type shows a slightly higher transfer rate in MiB/sec compared to the raw disk and Docker. This could be due to caching mechanisms or the way QCOW2 handles larger memory blocks.

2. **For 2gb ram and 4 core processor tests**

a. CPU Max Prime – 10000

1. **QEMU with raw disk**: The CPU executed an average of 21084.64 events per second.

2. **QEMU with QCOW2 disk**: Performance improved to an average of 21171 events per second.
3. **Docker with Ubuntu image**: The CPU performance is similar to QCOW2 with an average of 21155.74 events per second.

The performance increase from the raw disk to QCOW2 and Docker scenarios could be due to differences in disk I/O efficiency, as QCOW2 and Docker are designed to be more efficient with such operations

b. CPU Max Prime - 1000000

1. **QEMU with raw disk**:
    - Events per second: 41.27
    - Total number of events: 1982
    - Total time: 48.0256s
2. **QEMU with QCOW2 disk**:
    - Events per second: 41.27
    - Total number of events: 1982
    - Total time: 48.0243s
3. **Docker container**:
    - Events per second: 70.51
    - Total number of events: 3772
    - Total time: 48.0433s

From these results, it is evident that the Docker container is significantly faster in processing CPU-bound operations at this scale of the max prime number set to 1000000, compared to both QEMU environments with raw and QCOW2 disks

c. File I/O with file size 2Gb

1. **QEMU with raw disk**:
    - Write operations: 7632.09 per second
    - Throughput: 119.25 MiB/s
2. **QEMU with QCOW2 disk**:
    - Write operations: 6781.11 per second
    - Throughput: 105.95 MiB/s
3. **Docker container**:
    - Write operations: 11368.27 per second
    - Throughput: 176.98 MiB/s

In these tests, the Docker container shows the highest throughput and number of write operations per second, significantly outperforming both QEMU raw and QCOW2 disk images

d. File I/O with file size 4Gb

1. **QEMU with raw disk**:
    - Write operations: 8581.13 per second
    - Throughput: 133.77 MiB/s
2. **QEMU with QCOW2 disk**:
    - Write operations: 8495.74 per second
    - Throughput: 130.42 MiB/s
3. **Docker container**:

- Write operations: 8446.74 per second
- Throughput: 130.02 MiB/s

In these tests, the QEMU with a raw disk setup provided the highest throughput, indicating that the raw disk format might be more efficient for handling large file sizes compared to the QCOW2 format and Docker's layered filesystem

e. **Memory test with 16M memory block size**

1. **QEMU with raw disk**:
   - Total operations: 6400 (1872.43 per second)
   - Data transferred: 102400.00 MiB (29950.81 MiB/sec)
2. **QEMU with QCOW2 disk**:
   - Total operations: 6400 (1342.91 per second)
   - Data transferred: 102400.00 MiB (31085.53 MiB/sec)
3. **Docker container**:
   - Total operations: 6400 (2246.33 per second)
   - Data transferred: 102400.00 MiB (35941.33 MiB/sec)

From these results, the Docker container shows the highest throughput in terms of MiB/sec, followed by QEMU with a QCOW2 disk, and then QEMU with a raw disk. Docker's superior performance might be due to more efficient memory utilization or less overhead compared to full virtualization.

f. **Memory test with 32M memory block size**

1. **QEMU with raw disk**:
   - Total operations: 3200
   - Operations per second: 843.89
   - Data transferred: 102400.00 MiB
   - Transfer rate: 27004.53 MiB/sec
   - Total time: 3.7911s
2. **QEMU with QCOW2 disk**:
   - Total operations: 3200
   - Operations per second: 754.46
   - Data transferred: 102400.00 MiB
   - Transfer rate: 24144.64 MiB/sec
   - Total time: 4.2410s
3. **Docker container**:
   - Total operations: 3200
   - Operations per second: 657.54
   - Data transferred: 102400.00 MiB
   - Transfer rate: 23161.31 MiB/sec
   - Total time: 4.7292s

In this case, QEMU with a raw disk has the highest transfer rate and the shortest total time, indicating it performed the best in this particular memory test. QEMU with QCOW2 disk comes second in transfer rate.

3. **For 1gb ram and 2 core processor tests**

a. CPU Max Prime – 10000

1. **QEMU with raw disk**:
    - Events per second: 21332.32
2. **QEMU with QCOW2 disk**:
    - Events per second: 10816.78
3. **Docker container**:
    - Events per second: 21583.66

In these CPU tests, the Docker container achieved the highest number of events per second, indicating the best CPU performance among the three setups.

b. CPU Max Prime - 1000000

1. **QEMU with raw disk**:
    - Events per second: 41.40
    - Total time: 48.0395 seconds
    - Total number of events: 1980
2. **QEMU with QCOW2 disk**:
    - Events per second: 21.41
    - Total time: 48.0120 seconds
    - Total number of events: 1028
3. **QEMU with Docker container**:
    - Events per second: 41.12
    - Total time: 48.0895 seconds
    - Total number of events: 1974

In this scenario, it appears that the QEMU with raw disk and the Docker container have similar performance, with a slight edge for the raw disk in terms of events processed per second.

c. File I/O with file size 2Gb

1. **QEMU with raw disk**:
    - Write operations: 7722.86 per second
    - Throughput: 120.67 MiB/s
    - Total time: 10.008s
    - Total number of events: 176179
2. **QEMU with QCOW2 disk**:
    - Write operations: 7690.17 per second
    - Throughput: 120.16 MiB/s
    - Total time: 10.0125s
    - Total number of events: 175456
3. **Docker**:
    - Write operations: 11462.27 per second
    - Throughput: 178.95 MiB/s
    - Total time: 10.8679s
    - Total number of events: 257770

In this set of tests, the QEMU environment with the raw disk and QCOW2 disk has similar performance in terms of throughput, with a slight edge for the raw disk format in operations per second. However, the Docker container environment significantly outperforms both QEMU

environments in terms of operations per second and throughput, indicating more efficient file I/O operations.

d. File I/O with file size 4Gb

1. **QEMU with raw disk**:
    - Write operations: 16799.96 per second
    - Throughput: 260.62 MiB/s
    - Total number of events: 385180
    - Total time: 10.0035s
2. **QEMU with QCOW2 disk**:
    - Write operations: 10575.81 per second
    - Throughput: 165.22 MiB/s
    - Total number of events: 241031
    - Total time: 10.0148s
3. **Docker container**:
    - Write operations: 10594.84 per second
    - Throughput: 165.54 MiB/s
    - Total number of events: 241738
    - Total time: 10.0138s

In this set of tests for a 4GB file size, QEMU with a raw disk significantly outperforms the QCOW2 disk and Docker in terms of write operations per second and throughput. The raw disk's higher performance can be attributed to the absence of additional overhead associated with the QCOW2 format.

e. **Memory test with 16M memory block size**

1. **QEMU with raw disk**:
    - Total operations: 6400
    - Operations per second: 1844.00
    - Data transferred: 102400.00 MiB
    - Transfer rate: 29500.99 MiB/sec
    - Total time: 3.4702s
2. **QEMU with QCOW2 disk**:
    - Total operations: 6400
    - Operations per second: 1854.92
    - Data transferred: 102400.00 MiB
    - Transfer rate: 29578.75 MiB/sec
    - Total time: 3.4497s
3. **Docker container**:
    - Total operations: 6400
    - Operations per second: 2241.93
    - Data transferred: 102400.00 MiB
    - Transfer rate: 35870.88 MiB/sec
    - Total time: 2.8536s

The Docker environment outperforms both QEMU environments in terms of both operations per second and transfer rate, indicating a more efficient handling of memory operations.

f. **Memory test with 32M memory block size**

1. **QEMU with raw disk**:
    - Total operations: 3200
    - Operations per second: 843.77
    - Data transferred: 102400.00 MiB
    - Transfer rate: 27004.53 MiB/sec
    - Total time: 3.7911s
2. **QEMU with QCOW2 disk**:
    - Total operations: 3200
    - Operations per second: 754.46
    - Data transferred: 102400.00 MiB
    - Transfer rate: 24144.64 MiB/sec
    - Total time: 4.2410s
3. **Docker container**:
    - Total operations: 3200
    - Operations per second: 657.54
    - Data transferred: 102400.00 MiB
    - Transfer rate: 23161.31 MiB/sec
    - Total time: 4.7292s

The results indicate that QEMU with a raw disk delivers the highest performance in terms of both operations per second and transfer rate. QCOW2 shows slightly lower performance, which could be due to the overhead associated with the more complex features that the QCOW2 format supports.

4. **For 1gb ram and 4 core processor tests**

a. CPU Max Prime – 10000

1. **QEMU with raw disk**:
    - Events per second: 40213.38
2. **QEMU with QCOW2 disk**:
    - Events per second: 21121.70
3. **Docker container**:
    - Events per second: 21165.88

In these CPU tests, the QEMU with raw disk configuration achieved the highest number of events per second, indicating the best CPU performance among the three setups.

b. CPU Max Prime - 1000000

1. **QEMU with raw disk**:
    - Events per second: 41.30
2. **QEMU with QCOW2 disk**:
    - Events per second: 41.25
3. **Docker container**:
    - Events per second: 78.47

With the increase in computational complexity (increasing the prime numbers limit to 1,000,000), we see that the Docker container significantly outperforms both QEMU setups in terms of CPU events

processed per second. The Docker container's performance is nearly double that of the two QEMU setups, which have very similar results.

c. File I/O with file size 2Gb

1. **QEMU with raw disk**:
    - Reads per second: 6741.21
    - Writes per second: 6830.74
    - Throughput (MB/s) written: 105.33
2. **QEMU with QCOW2 disk**:
    - Reads per second: 6802.16
    - Writes per second: 8712.46
    - Throughput (MB/s) written: 106.28
3. **Docker container**:
    - Reads per second: 8144.47
    - Writes per second: 14421.01
    - Throughput (MB/s) written: 178.58

The Docker container setup shows a significantly higher file I/O performance, with more reads and writes per second and a higher throughput in MB/s compared to both QEMU configurations. The QEMU with QCOW2 disk shows slightly better performance than the raw disk setup, but both are substantially outperformed by the Docker container in this test**.**

d. File I/O with file size 4Gb

1. **QEMU with raw disk**:
    - Throughput written, MiB/s: 176.38
2. **QEMU with QCOW2 disk**:
    - Throughput written, MiB/s: 130.24
3. **Docker container**:
    - Throughput written, MiB/s: 178.55

The Docker container achieved the highest throughput, slightly outperforming the QEMU with raw disk setup, and significantly outperforming the QEMU with QCOW2 disk setup. This suggests that the Docker container environment may have a more efficient I/O path or less overhead in the I/O stack when writing large files.

e. **Memory test with 16M memory block size**

1. **QEMU with raw disk**:
    - Total MiB transferred: 29379.90 MiB/sec
2. **QEMU with QCOW2 disk**:
    - Total MiB transferred: 35842.13 MiB/sec
3. **Docker container**:
    - Total MiB transferred: 29407.92 MiB/sec

The QEMU with QCOW2 disk setup has the highest memory throughput, with 35842.13 MiB transferred per second, which is significantly higher than both the QEMU with raw disk and the

Docker container setups. This suggests that the QCOW2 setup is more efficient at handling large memory block transactions

f. **Memory test with 32M memory block size**

1. **QEMU with raw disk**:
   - Total MiB transferred: 72815.84 MiB/sec
   - Total time: 3.7658 seconds
2. **QEMU with QCOW2 disk**:
   - Total MiB transferred: 22470.45 MiB/sec
   - Total time: 4.5565 seconds
3. **Docker container**:
   - Total MiB transferred: 20406.28 MiB/sec
   - Total time: 5.0176 seconds

Given the above data, the QEMU with raw disk configuration not only transferred more data per second but also completed the operations in the shortest amount of time, making it the best performer in terms of both throughput and efficiency for the given memory test with a 32MB block size. This indicates that the QEMU with raw disk setup can handle large block memory operations more quickly and with higher throughput compared to the other configurations.

Shell Scripts of the Experiment

```
#!/bin/bash

iterations=5
#Running test on qemu for qcow image
run_test (){
local test_command="$1"
local test_name="$2"
echo "Running$test_name"
for ((i=1;i<=$iterations; i++)); do
echo "iteration $i:"
$test_command
echo     "'.....................................................................'."
done
}

Test Case 0: Limiting execution time to 30 sec
#run_test "sysbench --test=cpu --time=45 run" "Test Case 0:CPU time Limit"

#Test Case 1: Limiting max prime 100
run_test "sysbench --test=cpu --cpu-max-prime=100 --time=45 run" "Test Case 1: Cpu maxprime=100"

#Test Case 2: Limiting max prime to 1000
run_test "sysbench --test=cpu --cpu-max-prime=1000 --time=45 run" "Test Case 2: Cpu maxprime=1000"

#Test Case 3: Memory block size 16M
#run_test "sysbench memory --memory-block-size=16M run" "Test Case 3: Cpu memory block 16M"

#Test Case 4: Memory block size 8k
#run_test "sysbench memory --memory-block-size=8k run" "Test Case 4: Cpu memory block 8k"

#Test Case 5: File creating total size 1gb
#run_test "sysbench --num-threads=14 --test=fileio --file-total-size=1G --max-time=100 --file-test-mode=seqwr prepare" "Test Case 5: File io prepare"
#run_test "sysbench --num-threads=14 --test=fileio --file-total-size=1G --max-time=100 --file-test-mode=seqwr run" "Test Case 5: File io sequential R/W"
#run_test "sysbench --test=fileio --file-total-size=1G cleanup" "File Cleanup"


#Test Case 6: File creating total size 2gb
#run_test "sysbench --num-threads=14 --test=fileio --file-total-size=2G --max-time=100 --file-test-mode=rndrw prepare" "Test Case 6: File io prepare"
#run_test "sysbench --num-threads=14 --test=fileio --file-total-size=2G --max-time=100 --file-test-mode=rndrw run" "Test Case 6: File io random r/w run"
#run_test "sysbench --test=fileio --file-total-size=2G cleanup" "File Cleanup"
```

Automation Using Vagrant

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
Vagrant.configure("2") do |config|
config.vm.box = "ubuntu/hirsute64"
config.vm.provider "virtualbox" do |vb|
vb.memory = "1024"
vb.cpus = 2
end
#Below command will sync the local folder qemu
#to the /vagrant_folder folder on the virtual machine.
#Hence, any files added to the qemu folder on your host machine
# will be available in the /vagrant_folder folder on the virtual machine.
config.vm.synced_folder "qemu", "/vagrant_folder_vm"
config.vm.provision "shell", path: "vagrant_setup.sh"
end
```