COEN 241
Assignment : 1

Name: Steve Sebastian Dcosta
ID : W1606759

1. Ubuntu by using Windows subsystem for Linux

In this section of the report, we will go over the steps of how to install Linux using WSL and getting the Ubuntu GUI up and running for Windows 11 Operating System.

My Current System configurations are -

Device Name - GIGA_STEVE

Processor - 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 6 cores

RAM - 16 GB

System type - 64 bit

Downloading Ubuntu ISO

As the first step, it is essential that we download the correct ISO image for our system's configuration as we will be using this image for all our experiments. We have chosen Ubuntu 20.04 as our OS image which will be used by our Windows subsystem and Docker.

Ubuntu 20.04.1 LTS

After downloading it, place the image in the appropriate directory and remember the address of the directory as we will require it in the further steps.

a. Set up Windows and a Linux distro

I'll be using Ubuntu as it's the most popular and most widely supported OS for the most part. There are more tutorials and guides for getting out of sticky situations here than in any other distro.

To start, you need to ensure Windows can run WSL.

Hit Start, then search for and open "Windows Features".
You will need to make sure that both Windows Subsystem for Linux and Virtual Machine Platform are ticked. Hit OK and restart your computer if necessary.

Now, we need to set the WSL environment to version 2.

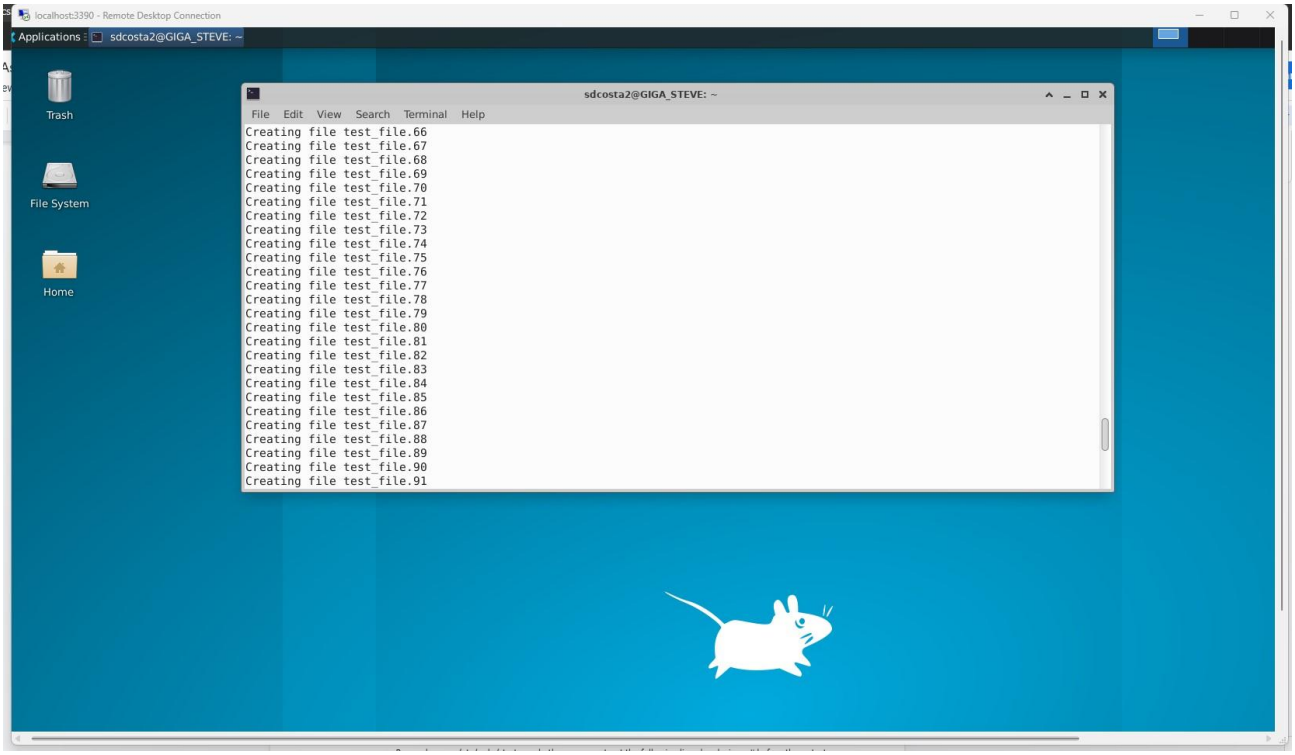
Open Powershell as administrator or Terminal on Windows 11. Then enter wsl --set-default-version 2. If prompted, download the WSL2 kernel and install it, then rerun the command.
Now we can install a Linux distro. I will use Ubuntu.

Open the Windows Store and search for "Ubuntu", or your distro of choice. This guide is specifically for Ubuntu or another Debian-based distro, so make sure you're modifying things where necessary if you're not.
While installing, you will need to set up an account and the rest. Click through doing whatever is nessecary.

b. Preparing Ubuntu

Now, update Ubuntu.

Run sudo apt-get update -y && sudo apt-get upgrade -y to update the OS.
If you run into an issue where nothing happens or time out, run sudo nano /etc/resolv.conf and change the nameserver x.x.x.x to nameserver 8.8.8.8 for Google DNS, or nameserver 1.1.1.1 for Cloudflare's DNS.



Verify that everything is up-to-date with lsb_release -a, or install Neofetch (sudo apt install neofetch) and run neofetch.

c. Set up the Desktop Environment

Now, for the exciting bit. We'll set up a desktop environment and use a remote access tool to interact with it. Remote Desktop is included with Windows. If you can install WSL you already meet the minimum version requirements (I think Windows Pro or higher is needed).

We'll use XFCE4, as it's the simplest.

Run sudo apt install -y xrdp xfce4 xfce4-goodies
Then, run the following:

sudo cp /etc/xrdp/xrdp.ini /etc/xrdp/xrdp.ini.bak
sudo sed -i 's/3389/3390/g' /etc/xrdp/xrdp.ini
sudo sed -i 's/max_bpp=32/#max_bpp=32\nmax_bpp=128/g' /etc/xrdp/xrdp.ini
sudo sed -i 's/xserverbpp=24/#xserverbpp=24\nxserverbpp=128/g' /etc/xrdp/xrdp.ini
echo xfce4-session > ~/.xsession

d. To finish setting up the RDP access:

Run sudo nano /etc/xrdp/startwm.sh, then comment out the following lines by placing a # before them: test -x /etc/x11... and the next line; echo /bin/sh /etc/x11...
Now, add the following on a new line at the very end: startxfce4
Save and close with Ctrl+S, Ctrl+X.
Start RDP with: sudo /etc/init.d/xrdp start.
Now, open Remote Desktop on your Windows host machine, and connect to localhost:3390.

2. Docker Installation and Important Instructions

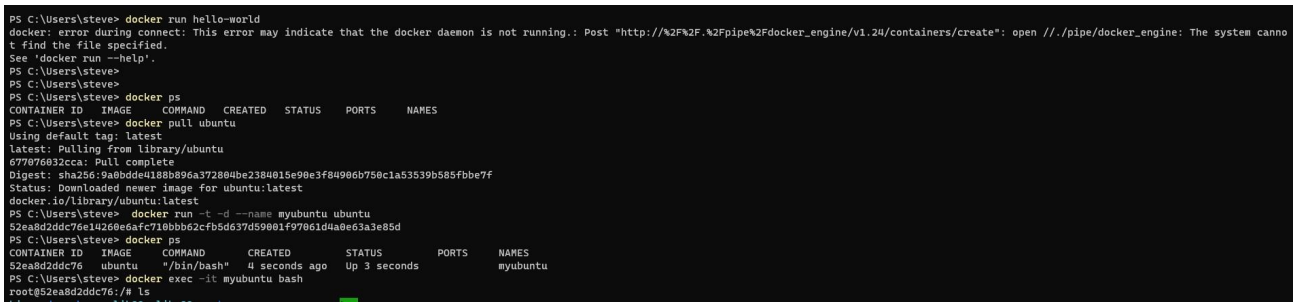
Install Docker desktop from the internet. Open Windows powershell and check this section of the report, we will be going over the installation steps for Docker, which is used to create

dW

Post installation, we can check if the docker has installed correctly on the system. To test docker, we simply type in the following command:

docker run hello-world

You should see the following output:



Thus, We have successfully installed Docker in our system.

NOTE: The following docker commands are useful for reference. These are as follows:

3. Experiments – Reports and Findings

In this section of the report, we will design and deal with test cases related particularly to sysbench CPU and File I/O commands. In this section, we will also test various configurations of the WSL to check if we do indeed get different results by changing the configurations of the VM.

Configuration 1 : 2GB Ram and 2 Cores

a. CPU TESTING

For CPU testing between WSL and Docker, we will use the following 3 test cases to check for performance. For our testing purposes, we will use the following sysbench command and the following test cases:

sysbench cpu --cpu-max-prime={some_value} --num-threads={some_value} --time= {some_value} run

1. max-prime = 2000 and time = 30 seconds
2. max-prime = 20,000 and time = 30 seconds
3. max-prime = 200,000 and time = 30 seconds

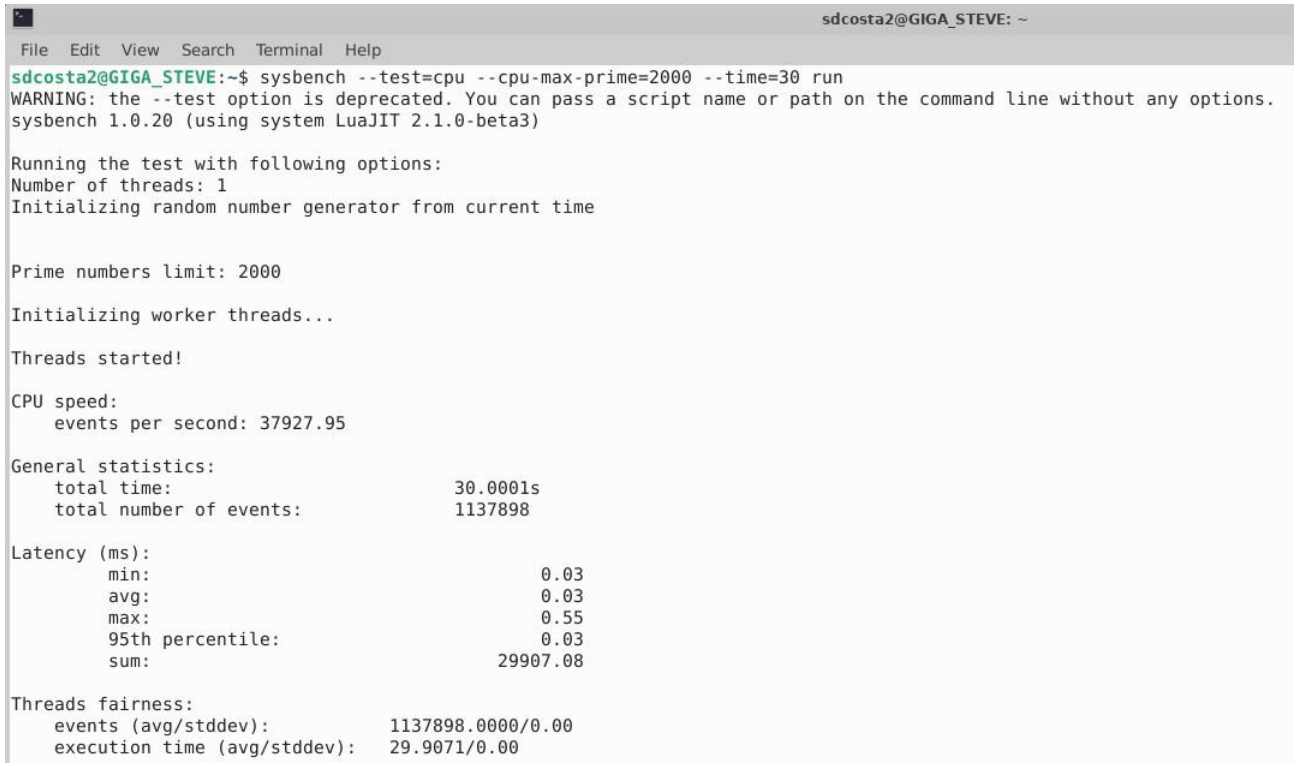
a. **WSL and Docker Test Results**

For our first test case, we choose a relatively smaller value of max prime numbers, in this case, 2000. We run the sysbench command 5 times to get accurate readings The screenshots of the execution and findings of the experiment are as follows :

NOTE: For execution of each of these commands, we use a bash shell script which automates the execution of commands. The bash shell script will be attached along with this report.

Running - ./WSL_2000_CPU.sh

iteration 1



A table detailing the result of our experiment is as follows:

Serial Number of Iteration	Events per second
1	37927.95
2	35785.37
3	36131.38
4	36447.29
5	36102.65

Some useful observations:

- Average events per second – 36478.928
- Maximum Number of events per second recorded – 37927.95
- Minimum Number of events per second recorded – 36102.65

Docker:

For the same test case, the result for docker are as follows:

```
root@52ea8d2ddc76:/# sysbench --test=cpu --cpu-max-prime=2000 --time=30 run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line witho
ut any options.
sysbench 1.0.20 (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: 2000

Initializing worker threads...

Threads started!

CPU speed:
    events per second: 36218.77

General statistics:
    total time:                          30.0001s
    total number of events:              1086620

Latency (ms):
         min:                                    0.03
         avg:                                    0.03
         max:                                    1.86
         95th percentile:                        0.03
         sum:                                29871.25

Threads fairness:
    events (avg/stddev):           1086620.0000/0.00
    execution time (avg/stddev):   29.8712/0.00
```

Iteration 1

The findings are reported in a table as follows:

Serial Number of Iteration	Number of events per second
1	36218.77
2	35963.78
3	36766.54
4	35773.94
5	36131.63

Some useful observations:

- Maximum events/second reading recorded - 36766.54
- Minimum events/second reading recorded – 35773.94
- Average events/second – 36170.932

Conclusion – We note that according to our experiment, WSL has similar speeds to Docker.

TEST CASE 2: WSL AND DOCKER:

For our test case 2, we increase the max-prime argument value to 20,000, which is 10 times the first test case value and we keep the time parameter value constant to maintain consistency between results. Now we will check if the increase in this parameter value impacts our system performance.

WSL:

The execution of the test case sysbench command is as follows:

```
sdcosta2@GIGA_STEVE:~$ sysbench --test=cpu --cpu-max-prime=20000 --time=30 run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line v
sysbench 1.0.20 (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: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:  1397.20

General statistics:
    total time:                          30.0007s
    total number of events:              41919

Latency (ms):
         min:                                    0.67
         avg:                                    0.72
         max:                                    4.70
         95th percentile:                        0.75
         sum:                                29980.12

Threads fairness:
    events (avg/stddev):           41919.0000/0.00
    execution time (avg/stddev):   29.9801/0.00
```

Iteration 1

Upon running the command for 5 times, the findings are as follows:

Serial Number of iteration	Events/Second
1	1397.2
2	1395.21
3	1401.11
4	1415.86
5	1384.21

Some useful observations:

- Maximum number of events per second recorded – 1415.86
- Minimum number of events per second recorded – 1384.21
- Average number of events per second recorded – 1398.718

DOCKER:

Docker execution of the test case sysbench command is as follows :

```
root@52ea8d2ddc76:/# sysbench --test=cpu --cpu-max-prime=20000 --time=30 run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line witho
ut any options.
sysbench 1.0.20 (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: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:  1447.82

General statistics:
    total time:                          30.0008s
    total number of events:              43438

Latency (ms):
         min:                                    0.67
         avg:                                    0.69
         max:                                    1.31
         95th percentile:                        0.74
         sum:                                29986.62

Threads fairness:
    events (avg/stddev):           43438.0000/0.00
    execution time (avg/stddev):   29.9866/0.00
```

Iteration 1

Upon running the command for 5 times, the findings are as follows:

Serial Number of Iteration	Number of Events/Second
1	1447.82
2	1431.82
3	1416.03
4	1433.85
5	

Some useful observations:

- Maximum number of events per second recorded –1447.82
- Minimum number of events per second recorded – 1416.03
- Average number of events per second – 1431.152

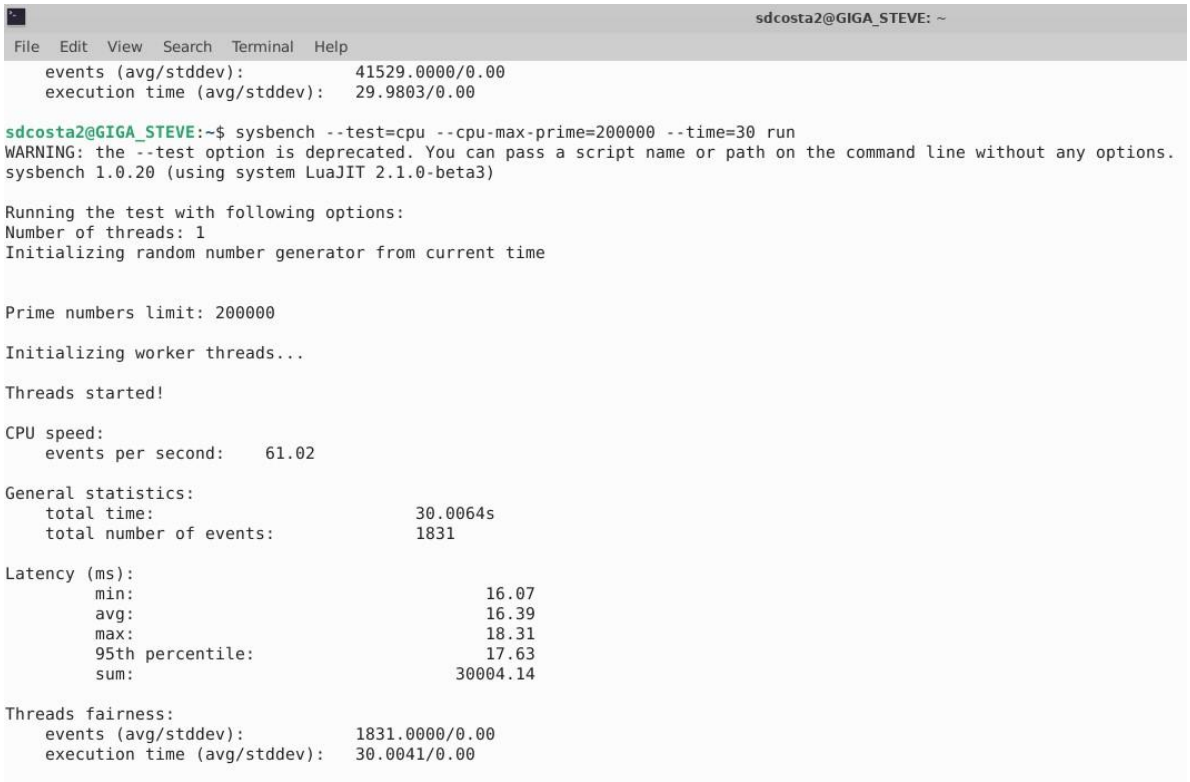
Conclusion: We see that as we increase the max-prime argument value, we see the number of events per second decrease for our current experimental configuration. Docker is slightly faster than WSL on Windows.

Test Case 3: WSL AND DOCKER

For our test case 3, we increase the cpu-max-prime argument value to 200,000 while keeping the time parameter constant for result consistency. Now, we again run the test case sysbench commands for both WSL and docker.

WSL:

The execution of test case sysbench command is shown as follows:



Iteration 1

Upon running the command 5 times, the findings are as follows:

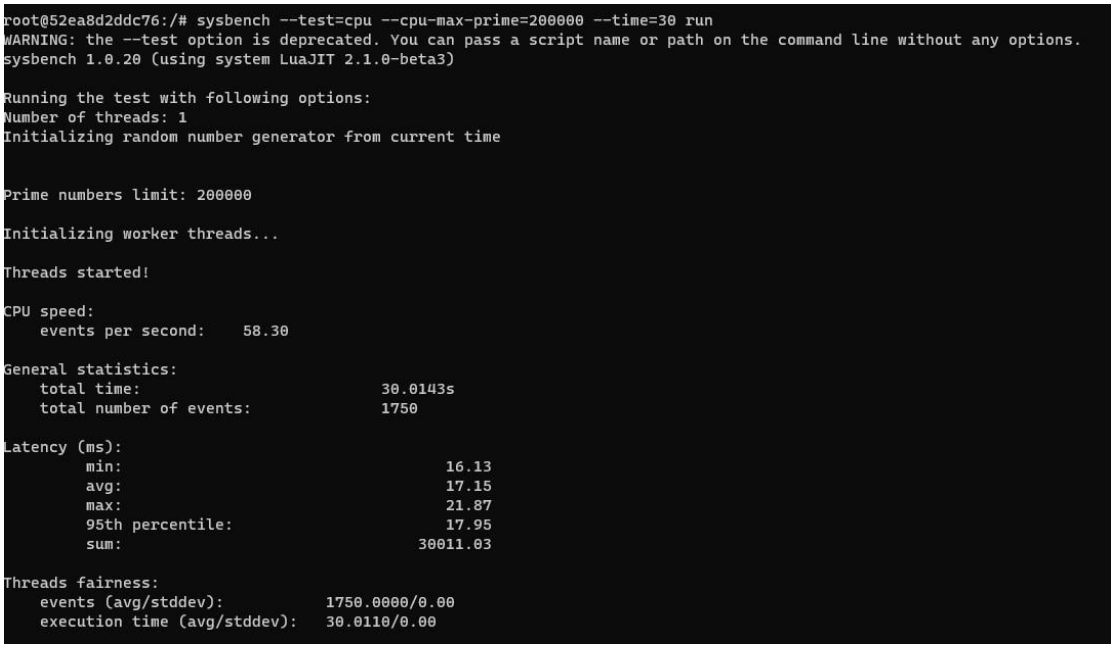
Serial Number of Iteration	Number of Events per Second
1	61.02
2	60.56
3	63.22
4	60.91
5	61.11

Some useful observations:

- Maximum number of events per second recorded – 63.22
- Minimum number of events per second recorded – 60.56
- Average number of events per second – 61.364

DOCKER:

The test case sysbench command execution for docker is as follows:



Iteration 1

Upon running the command 5 times, the findings are as follow

Serial Number of Iteration	Number of Events per Second
1	59.26
2	58.3
3	58.6
4	59.13
5	58.90

Some useful observations:

- Maximum number of events per second recorded – 59.26
- Minimum number of events per second recorded – 58.3
- Average number of events per second – 58.838

Conclusion: We note that the number of events further decrease as we continue to increase the cpu-max-prime argument value. WSL and Docker values continue to be quite similar.

Based on our findings, we can safely conclude that as we increase the cpu-max-prime argument value, our number of events per second continue to decrease, as in, greater the value of cpu-max-prime argument, lower the number of events per second for sysbench CPU testing.

b. FILE I/O TESTING (WSL AND DOCKER):

For File I/O testing, I will use two modes which sysbench supports, which are:

i. Sequential Rewrite (seqrewr)

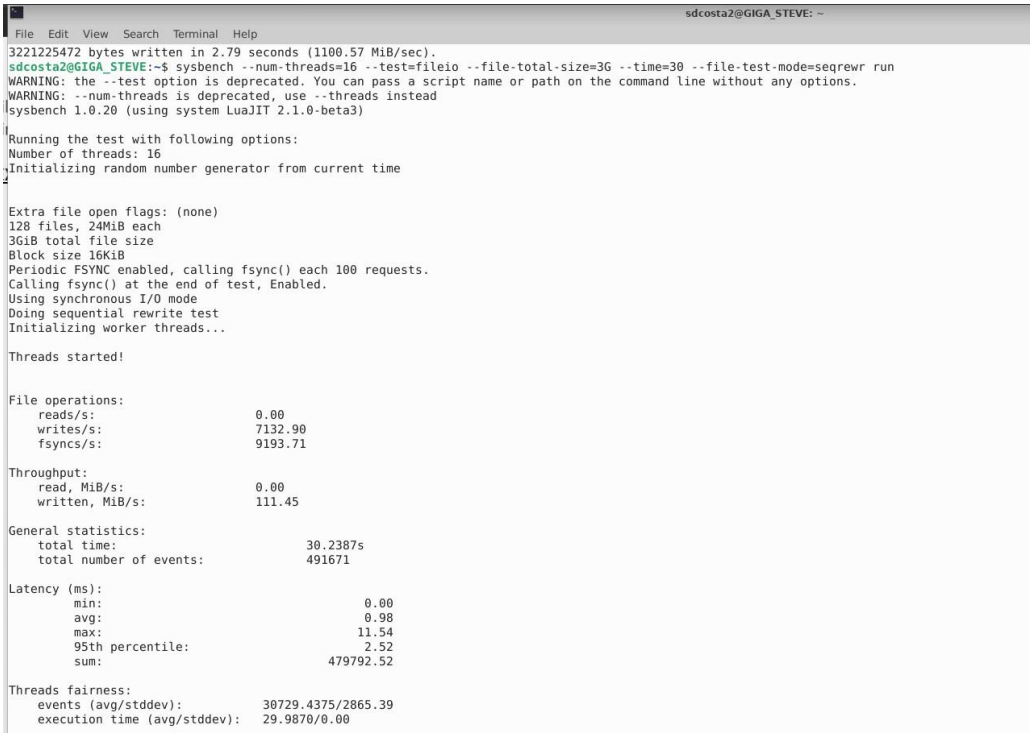
ii. Combined random read/write (rndrw)

We will keep the file size constant at 3GB and will test the performance against our experimental setups accordingly.

WSL EXECUTION:

i. Sequential Rewrite:

The execution for WSL sequential rewrite command is shown as follows:



After executing the command 5 times, We get the following results:

Serial number of Iteration	Results
1	reads/s = 0 writes/sec = 7132.9 fsyncs/sec = 9193.71 events/second = 16259.66
2	reads/s = 0 writes/sec = 7212.9 fsyncs/sec = 9121.71 events/second = 16249.61
3	reads/s = 0 writes/sec = 7298.9 fsyncs/sec = 9179.71 events/second = 16219.43
4	reads/s = 0 writes/sec = 7189.9 fsyncs/sec = 9221.71 events/second = 16266.21
5	reads/s = 0 writes/sec = 7152.9 fsyncs/sec = 9151.71 events/second = 16182.01

ii. **<u>Combined Random Read Write (rndrw)</u>**

The execution of the rndrw command is shown below as follows:

```
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 16
Initializing random number generator from current time


Extra file open flags: (none)
128 files, 24MiB each
3GiB 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 r/w test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      4050.62
    writes/s:                     2700.36
    fsyncs/s:                     8708.56

Throughput:
    read, MiB/s:                  63.29
    written, MiB/s:               42.19

General statistics:
    total time:                          30.2310s
    total number of events:              465335

Latency (ms):
         min:                                    0.00
         avg:                                    1.03
         max:                                  429.94
         95th percentile:                        2.81
         sum:                               479760.87

Threads fairness:
```

Iteration 1

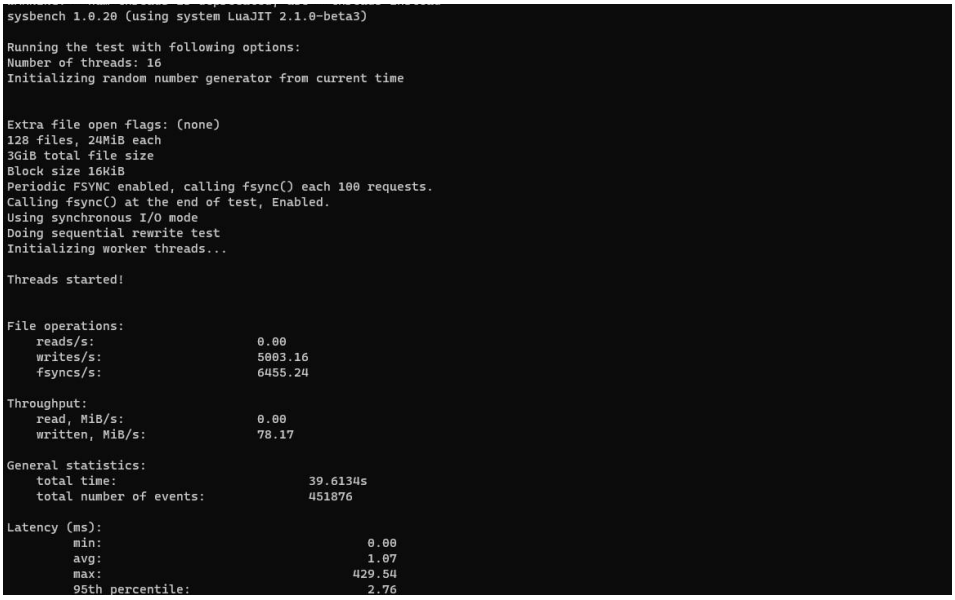
Upon running the command 5 times, we get the following results:

Serial Number of Iteration	Results
1	reads/s = 4050.62 writes/s = 2700.36 fsyncs/s = 8707.56 events/second = 15329.64
2	reads/s = 4061.13 writes/s = 2717.31 fsyncs/s = 8733.79 events/second = 15145.3
3	reads/s = 4103.87 writes/s = 7096.89 fsyncs/s =8719.05 events/second = 15342.13
4	reads/s = 4039.72 writes/s = 2687.37 fsyncs/s = 8713.81 events/seconds = 14842.03
5	reads/s = 4491.06 writes/s = 2327.07 fsyncs/s = 8713.69 events/seconds = 15486.16

DOCKER EXECUTION:

The docker execution is given as follows:

i. Sequential Read Write



Iteration 1

Running the command for 5 times, we get the following readings:

Serial number of Iteration	Results
1	reads/s = 0 writes/sec = 5003.16 fsyncs/sec = 6455.24 events/second = 11407.1
2	reads/s = 0 writes/s=5435.34 fsyncs/sec = 6342.32 events/second = 11343.24
3	reads/s = 0 writes/s = 5958.73 fsyncs/sec = 6684.54 events/second = 11846.34
4	reads/s = 0 writes/s = 5091.93 fsyncs/s = 6321.54 events/second = 11654.18
5	reads/s = 0 writes/s = 5546.54 fsyncs/s = 6323.20 events/second = 11234.32

ii. Combined Random Read Write

The execution for combined random read write command is as follows:

```
Running the test with following options:
Number of threads: 16
Initializing random number generator from current time


Extra file open flags: (none)
128 files, 24MiB each
3GiB 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 r/w test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      4010.04
    writes/s:                     2673.36
    fsyncs/s:                     8618.54

Throughput:
    read, MiB/s:                  62.66
    written, MiB/s:               41.77

General statistics:
    total time:                          30.2226s
    total number of events:              460440

Latency (ms):
         min:                                    0.00
         avg:                                    1.04
```

Iteration 1

Upon running the command 5 times, we have the following results:

Serial number of Iteration	Results
1	reads/s = 4010.04 writes/sec = 2673.36 fsyncs/sec = 8618.54 events/second = 15234.95
2	reads/s = 4155.49 writes/s=2769.99 fsyncs/sec = 8129.15 events/second = 15019.5
3	reads/s = 3353.95 writes/s = 2569.02 fsyncs/sec = 8688.31 events/second = 15574.26
4	reads/s = 4674.66 writes/s = 2782.83 fsyncs/s = 8370.93 events/second = 15796.33
5	reads/s = 4488.90 writes/s = 2659.14 fsyncs/s = 8975.53 events/second = 15094.1

5. Github Repository Information

Account Name – steve261998
Repository - Cloud_Computing
Folder which contains the assignment – Homework 1
Link to Repository -https://github.com/steve261998/Cloud_Computing