

# COEN 241 HW 1

## System vs OS Virtualization

Due: Monday, October 11th 2021, at 7:00 PM

---

### Learning Objectives

The aims of the assignment include the following learning objectives:

- Understanding the real-world use case of two virtualization techniques / technologies:
  - System Virtualization via QEMU
  - OS Virtualization via Docker
- Understand the operations related to each technologies
- Understand and demonstrate the performance differences of each technology using benchmarks and finally compare the results in a comprehensive report.

### Environment Setup

You are free to use your personal computer for this assignment. The set of requirements for using your personal computers is as follows:

- Intel CPU with at least 2 cores or a Mac M1
- 4 GB memory
- 20 GB free disk space
- OS: Windows, Linux (Ubuntu preferred) or Mac OS X

If you are using a personal computer that does not support such requirements, please do reach out to me, so that we can discuss some options.

### System Virtualization (QEMU) Setup

#### What is QEMU?

QEMU is a free and open-source hypervisor. It emulates the machine's processor through dynamic binary translation and provides a set of different hardware and device models for the machine, enabling it to run a variety of guest operating systems. It is more lightweight than virtualbox and allows near-native speed (when using QEMU with KVM on Linux). QEMU supports various types of CPUs, such as Intel and ARM, and also supports integrations with various host operating systems. It is less user-friendly and requires more effort to setup.

# Installing QEMU

## For All Operating Systems

For this assignment, you will be installing a Ubuntu guest virtual machine. Thus you must download the appropriate Ubuntu 20.04 server ISO image from one of the links below:

- Intel CPU: [Ubuntu 20.04 03 Server](#)
- ARM (M1): [Ubuntu 20.04\\_3 Server for ARM](#)

From here, the installation instruction varies between which OS you are running.

## Linux Installation instructions

QEMU is the easiest to install if you have Ubuntu. You can install qemu (under Ubuntu) simply by typing the following command in the terminal:

```
$sudo apt-get install qemu
```

You can then create the QEMU Image by running the following command.

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

Given the image, install the VM using the command below (which takes the iso file as a “cdrom” and the qemu image as a “hard disk”):

```
$sudo qemu-system-x86_64 -hda ubuntu.img -boot d -cdrom  
./[UBUNTU_SERVER_ISO_FILE_NAME] -m 1536
```

For each of these commands, please refer to the QEMU manual for more details [1].

## Mac OS X (Intel) Installation instructions

The instructions to install QEMU on the Intel Macs can be found in the following link <https://wiki.qemu.org/Hosts/Mac>. I suggest using the homebrew method. You can find homebrew installation instructions in [2], if it is not already installed in your machine. Once QEMU is installed, you can follow a similar procedure as the Linux installation to create a QEMU image and install the VM.

## Mac OS X (M1) Installation instructions

Given that M1 is a very new processor, there are not many resources available to simply install QEMU with a single command line. One resource I have found is the following link <https://gist.github.com/citruz/9896cd6fb63288ac95f81716756cb9aa>

Note that the VM image file names may be different based on which version you have downloaded.

## Windows Installation Instructions

Running QEMU on Windows is **NOT RECOMMENDED**, as most servers are Linux/Unix based and most open source virtualization works best with Unix based OSes. However, it is certainly possible. Here is an article that describes the entire process in detail in [3], which requires you to be able to change some Window environments variables.

# OS Virtualization (Docker) Setup

## What is Docker?

Docker is a set of platforms that enable OS-level virtualization to deliver software in packages called containers. Docker is the most popular container management platform.

## Installing Docker

While we will be installing Docker Desktop (for Windows and Mac), we will be interacting mostly with Docker CLI. Docker Desktop installation will include Docker Engine, Docker CLI client, Docker Compose and more. You can learn more about the Docker CLI in [5]. I have included the corresponding links to find the installation instructions for each type of OSes. Installing Docker should be much simpler than installing QEMU.

### Linux Installation instructions

Note: Ubuntu does not come with Docker Desktop, so we would just need to install the Docker Engine and the Docker CLI separately.

- <https://docs.docker.com/engine/install/ubuntu/>

### Mac OS X Installation instructions

- Intel CPU: <https://docs.docker.com/desktop/mac/install/>
- ARM (M1):: <https://docs.docker.com/desktop/mac/apple-silicon/>

### Windows Installation instructions

- <https://docs.docker.com/desktop/windows/install/>

## Installing sysbench

Sysbench[4] is an open-source and multi-purpose benchmark utility that evaluates the parameter features tests for CPU, memory, I/O and much more. We will use sysbench to understand the performance characteristics of each of the virtualization technologies.

## For Ubuntu Guest VM

Sysbench installation on Ubuntu is very simple. Just run the following commands:

```
$sudo apt update
$sudo apt install sysbench
```

Note: Do not be confused with the password for the guest VM.

## For Docker

Please use the Docker image called `csmnpp/ubuntu-sysbench`, which has sysbench preinstalled. You can find the image in <https://hub.docker.com/r/csmnpp/ubuntu-sysbench>. Once you have found the image, you would need to learn how to interact with a container. Please refer to the demo in the lecture. You can also refer to the tutorials in [6], [7].

## What to Experiment?

Given the sysbench tool, we can have some basic ideas about the system performance by running it against the system under test. In assignment, we will focus on the `cpu` and `fileio` test modes (Section 4.1 and 4.5 in [4]) of Sysbench. The following instructions present some ideas about how to conduct a meaningful measurement. Please read them carefully, and then produce the measurement results. If the results are incorrectly generated and/or can not be well justified, you will not be able to get a full credit for this assignment.

- Test different arguments of `qemu`
  - Some examples: `-m`, `-smp`, `-accel` and more
  - Be sure to document your experiments and why you used them.
- You need to figure out all the test cases (under the `cpu` and `fileio` test modes).
- For each test case, you need to figure out the “right” parameters. Some examples are,
  1. Choose the parameter for “`-cpu-max-prime`” to ensure your test case won’t end in a short time period (too small) or will never end (too large).
  2. Decide the file size (e.g., `-file-total-size`) for Sysbench `fileio` test mode.
- Usually, it’s reasonable to make a test case lasting for at least 30 seconds, but for such test cases, please justify your configurations for doing so.
- You need to repeat the sysbench measurement **at least 5 times** for each test case, and report the **average, min, max and std.** values of your results (the Sysbench will report some user-level performance data, e.g., total time).
- To reduce test variation, you need to test each case under a similar test environment.

- For the fileio test, after one test, the operating system will cache the accessed files. If we don't manually drop such cache, the following tests will finish much faster, as most I/O data will be served directly from the 1 memory cache instead of disks. You have to manually drop such cache in the Host using the commands (in superuser or administrator mode):
  - For Linux:
 

```
echo3 > /proc/sys/vm/dropcaches
```
  - For Mac OS:
 

```
sync && sudo purge
```
  - For Windows (Windows, this is not readily available, thus you must download the tool from [here](#)):
 

```
sync
```
- For each of the experimental runs, **create a bash script to automate the experiment.**
- Please well organize your experimental setup, configurations, and data (using figures and/or tables); and report them.

## Submission Instruction

Create a submission for the assignment on the Camino containing the following two pieces of information:

1. a URL such that the command 'git clone URL' would download your repository (assuming that the repository has been shared with the requesting user);
2. a git commit ID within your repository that you want markers to assess that contains the following.
  - a. The project report in PDF format
  - b. Shell scripts to run your experiments

## Repository Requirements

- You are expected to work on your assignment within a git repository. The easiest way to acquire URLs that reach your repository is to push your work to a (free) cloud-based git service, such as GitHub, Bitbucket, GitLab, or Altitude (a GitLab server run locally within the Department of Computer Science that you can access). The aforementioned cloud services allow you to create private repositories.
- You must invite teaching staff to collaborate on your repository, so that they can access the URL that you submit. Please send an invitation to [schoi3@scu.edu](mailto:schoi3@scu.edu)
- Do not apply git history-rewriting operations to your repository: it is important that the markers can see how your commits completed your assignment, and how you worked on the assignment over time.

# Rubric / What to Write on the Report

- Your report should be a single PDF file included in the github repository. Most of the grade on this assignment is based on this report.
- You should not only follow the instructions to do your assignment, but also fully understand what you are doing
- The report should include (but not limited to):
  - Detailed configurations (CPU, Mem, etc...) of your experimental setup: **5 points**
  - Present main steps to enable a QEMU VM. In addition, please present the detailed QEMU commands, and VM configurations: **10 points**
  - Present main steps to enable a Docker container. In addition, please describe the operations you use to manage Docker containers (and some other operations which you think are also important): **10 points**
  - Proof of experiment. Include screen snapshots of your Docker and QEMU running environments for each experiment: **5 points**
  - Present how you conduct your measurements in three different scenarios for each virtualization technology: **20 points**
  - Shell scripts for running the experiment: **20 points**
  - Present how you use performance tools to collect performance data. For CPU utilization, you should at least divide them into two parts including user-level and kernel-level. For I/O, you should present I/O throughput, latency, and disk utilization: **10 points**
  - Presentation and analysis of the performance data:: **10 points**
  - Understandability/Neatness: **10 points**
  - Git Repository Information: **5 points**
  - Extra Credit: Provide the correct Vagrant and Docker files for your VM and Docker Container. **30 points**

## About the Extra Credit

For cloud deployment, automation is critical. Thus there are tools that enable developers to easily store the configurations of their VMs & Containers and launch them on the cloud. One such tool is Vagrant and the other is Docker File. Writing the configurations for these platforms requires practice and may take multiple tries before getting them right. However, if you are ambitious, you can try to write both Vagrant and Docker File for this assignment for future uses (and be rewarded hefty for it!). Note that Vagrant was created to be used with VirtualBox, so it may be tricky to run and debug the Vagrant file with QEMU. My suggestion is to do the assignment using QEMU, but build the Vagrant file using Virtualbox. If you are ambitious, you can try running Vagrant with QEMU as well! However, note that the extra credit is the same regardless of which hypervisor you decide to test the Vagrant file on.

Here are some references to help you write Vagrant and Docker Files

- Writing Vagrant File <https://www.vagrantup.com/docs/vagrantfile>

- Writing Docker File  
[https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)

For those of you who want to run Vagrant with QEMU, this would be the first step to try.

- <https://computingforgeeks.com/using-vagrant-with-libvirt-on-linux/>

## References

- [1] QEMU Documentation: <https://qemu.weilnetz.de/doc/6.0/>
- [2] Homebrew: <https://brew.sh/>
- [3] How do I run QEMU on Windows: <https://linuxhint.com/qemu-windows/>
- [4] Sysbench manual: <https://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf>
- [5] Docker CLI Reference: <https://docs.docker.com/engine/reference/commandline/cli/>
- [6] Docker exec: <https://docs.docker.com/engine/reference/commandline/exec/>
- [7] Docker Tutorials: <https://docker-curriculum.com/>