Create Simple Container,

but it was not simple …

Timur Kharin

Total Virtualization

Innopolis University

2024

# Intro

The idea behind this project is to create a virtualized environment that isolates system resources for running containers. This is achieved through the use of Linux namespaces and cgroups

## Features

* The virtualized environment is created using namespaces and cgroups
* The environment is designed to run containers
* Tries to make it as CLI command

## Links

<https://github.com/timur-harin/virtual_timur.git>

# Tests

To test the container against other products and the host machine, we have measured the following performance metrics:

**CPU:** We measured the total CPU time used by the container and compared it to the host machine. This can be used to measure the efficiency of the container in utilizing CPU resources

**Memory:** We measured the memory usage of the container and compared it to the host machine. This can be used to measure the container's memory footprint and performance

**File I/O:** We measured the read and write speeds of the container's filesystem and compared them to the host machine. This can be used to measure the performance of the container's file I/O operations

These metrics are primary measurable specs of a standard PC and were stated in the lab assignment

## commands

| Metric | Sysbench command | Why this command | What is interesting in sysbench output |
| --- | --- | --- | --- |
| CPU total time [sec] | sysbench cpu  --threads=100  --time=60 | This command loads all CPU cores heavily by creating 100 threads that perform prime number computations. This workload saturates the CPU cores with computational tasks that cannot be significantly optimized from the hardware side | total time |
| threads | sysbench  threads  --threads=64  --thread-yields  =100  --thread-locks=  2 run | This command tests the performance of the system's threading capabilities by creating 64 threads and configuring various thread-related options | total № of events, events avg and stddev |
| memory concurrent write test | sysbench memory  --threads=100  --time=60  --memory-oper=w  rite run | This command tests the memory write access in a concurrent environment. The purpose of this test is to measure the performance of the system's memory subsystem under high write access and to evaluate the effectiveness of paging | Memory speed |
| memory stress test | sysbench memory  --memory-block-  size=1M  --memory-total-  size=10G run | This command will simulate a high memory stress condition by performing a large number of write operations to memory | Memory speed |
| fileio write test | sysbench fileio  --file-total-si  ze=10G  --file-test-mod  e=rndrw  --time=120  --time=300  --max-requests=  0 run | This command will test the write performance of the file I/O system by performing random write operations on a file of size 10GB | Ops/sec (read, write, fsyncs), latency |

# 

# Table With Metrics

Let’s see all reports from the telegram message <https://t.me/c/2010893534/139>

So thank you so much Dmitry Alekhine

And Darko was agreed

<https://t.me/c/2010893534/141>

So let’s see final tables here:

1. Host: <https://github.com/timur-harin/virtual_timur/blob/main/report_host.md>
2. Container: <https://github.com/timur-harin/virtual_timur/blob/main/report_container.md>

# 

# Explanation Why Metrics Differ

The variation observed in all the metrics mentioned above compared to the benchmark results on the host machine can be attributed to loop device isolation. This setup involves executing file I/O system calls to a loop device first and then to a hard drive (which houses an image file) subsequently. However, due to the absence of CPU and memory constraints in the default cgroups settings, the respective tests exhibit nearly identical performance when compared to the host machine

## File IO test

The performance evaluation of the Proof of Concept (POC) revealed that the operations per second (ops/s) for read, write, and fsync operations were approximately 2-3 times slower compared to the host machine. Moreover, the average latency was found to be around 4 times higher than that of the host, while the maximum latency was approximately 20 times greater than observed on the host machine

# Sources

1. [SysBenchExample] - “How to Benchmark Your System (CPU, File IO, MySQL) with Sysbench”  
   <https://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench>
2. [HowCgroupsWorks] - “Containerization mechanisms: cgroups” on Russian

<https://habr.com/ru/companies/selectel/articles/303190/>

1. [WhatIsLoopDevice] - “What is Loop Device” on Russian

<https://dzen.ru/a/YzRrCm81Q2dhE2FM>

# Main script

[timur.sh](https://github.com/timur-harin/virtual_timur/blob/main/timur.sh) - all comments inside it

**Main Steps:**

1. Create Container: Isolate PID, mount namespace, and network namespace.
2. Put Container Filesystem in a File: Use loop device to create a file with the container filesystem and try creating a file in the new mount point.
3. Benchmark Container vs Host (LXC): Use sysbench to benchmark CPU total time, FileIO read/write, Memory access, and Thread execution.
4. Write Report: Explain the created container, steps performed, reasons behind choices, and provide insights from benchmarking.

**Technology:** Using Linux namespaces and tools like unshare to create a container-like environment, sysbench for benchmarking, and scripting in Bash to automate the setup and benchmarking process. Docker-like features are achieved using low-level Linux facilities, making it a hands-on learning experience in containerization and benchmarking.

# My tries for CLI

[virtual\_timur.sh](https://github.com/timur-harin/virtual_timur/blob/main/virtual_timur.bash)

This script provides a command-line interface (CLI) for managing a virtual environment called "Virtual Timur". The script was intended to be used as a CLI tool, but due to time constraints, it is not fully functional

Here's a summary of what this script does:

* It checks if the script is run with root privileges. If not, it displays an error message and exits.
* It defines several variables that specify the paths for the Virtual Timur project, images, and mounts.
* It defines a function called show\_help() that displays a help message with instructions on how to use the script.
* It defines a function called init\_virtual\_timur() that initializes the Virtual Timur environment. This function creates necessary directories, downloads a base rootfs image, and sets up an isolated network interface.
* It defines a function called get\_default\_image() that downloads the base rootfs image if it doesn't already exist.
* It defines a function called setup\_isolated\_iface() that sets up an isolated network interface for the Virtual Timur environment.
* It defines a function called remove\_isolated\_iface() that removes the isolated network interface.
* It defines a function called clear\_virtual\_timur() that clears all Virtual Timur instances by removing all containers, clearing the project directory, and removing the isolated network interface.
* It defines a function called build\_virtual\_timur() that builds a new container for the Virtual Timur environment. This function creates a loopback device, formats it with ext4, mounts it, and extracts the base rootfs image into it.
* It defines a function called run\_virtual\_timur() that runs a container for the Virtual Timur environment. This function creates a control group (cgroup) for CPU and memory management, and executes a chroot environment with mounted filesystems and a Bash shell.