Storage Systems (StoSys) Course Project Handbook

(XM_0092)
P2 (November - December), 2021
Version: 2.0

Animesh Trivedi (a.trivedi@vu.nl)
Matthijs Jansen (m.s.jansen@vu.nl)
Sacheendra Talluri (s.talluri@vu.nl)

Storage Systems (StoSys) Course Project Handbook 1

Revision Information 3

0. Code of Conduct (Plagiarism) 3

1. Overview 4

2. Logistics 4

2.1 What is given to you 5

2.2 How to ask for help when stuck in the project 6

2.3 Late/Slip days 6

2.4 Group Management Logistics 7

3. Assessment Criteria 7

3.1 Grading guidelines 7

3.2 Notes on partial assessment for Milestones 8

4. StoSys Project Work 9

4.1 Milestone 1 (week 1) : A new device is in town 10

4.2 Milestone 2 (week 2) : I can’t read, is there a translator here? 22

4.3 Milestone 3 (week 3) : It’s 2021, we recycle 25

4.4 Milestone 4 (week 5) : We love Rock(sDB) ‘n’ Roll! 27

4.5 Milestone 5 (week 7) : Wake up, Neo 32

5. Bonus Ideas and Timeline 33

Appendix 35

A. Step by step guide for setting up Ubuntu installation 35

B. Setup of VNC through Remmina 36

C. Running inside Virtualbox, VirtualMachineManager or VMWare 37

Revision Information

- November 1st, 2021: version 2.0 is released with NVMe ZNS devices and RocksDB. OC-SSD based experiments are removed.
- October 25th, 2020: version 1.1 with some corrections from Corné Lukken
- October 25th, 2020: version 1.0

0. Code of Conduct (Plagiarism)

We have a ZERO-TOLERANCE cheating, fraud, and plagiarism policy.

You are encouraged to discuss specific issues with other students, search online, read Linux/Storage code, and share knowledge with other fellow students. After all, the whole field of computer science is built with open-collaboration. ***HOWEVER, in any circumstances you are expected to write your own code***. ***Do not buy or sell code online***. **Do not copy code. Do not give out code to other students.** It is quite unlikely that a project of this scope will result in two students/groups writing almost identical code. Any similarities in the code, or style will lead to further investigation, with a possibility of reporting to the examination board - an outcome we both would like to avoid.

1. Overview

These are very exciting times to do storage research and the availability of many flexible, easy-to-modify storage stacks gives you an opportunity to think about how a modern, high-performance, storage stack should look like. Why now? For the last 50-60 years we have been happy with the hard disk drives (HDDs), with the block layer and associated file system inside the Linux kernel. However, this basic design was constrained by the performance of the rotating magnetic HDDs. With the advent of modern non-volatile memory (NVM) storage technology the storage game has changed completely. These new storage devices (such as NAND flash Solid State Drives, SSDs - a type of NVM storage) offer almost 2-3 orders of magnitude performance gains. At the same time these SSD devices internally are complex, thus offering a variety of tradeoffs in performance, energy, cost, and reliability. There is no one storage stack design that can fit all. In this project, we will build one from scratch without being hindered by the legacy block and file interfaces.

The objective of this course project is to develop a working storage stack in user space. More specifically you will need to

1. Get familiar with Zone Namespace (ZNS) devices and their capabilities
2. Implement a Flash Translation Layer (FTL) with garbage collection (GC)
3. Implement an application-specific file system on top of the FTL and integrate it into a production-quality KV store, RocksDB
4. Make the device FTL, and file system persistent

These concepts are covered in the due time in the lecture. Hence, the idea here is to give you conceptual as well as practical knowledge regarding how an SSD works with an state-of-the-art flash SSD interface, i.e., ZNS. ZNS specification is ratified and merged in the NVMe specification 2.0 in August, 2021.

Furthermore, building such storage stacks in the user-space is not a “hack” just to demonstrate something. As Moore’s Law is slowing down, user-space specialization of storage stacks is one of the most promising and popular ways of delivering performance to modern data-intensive applications such as BigData processing, graph computation, Bioinformatics applications, etc. There are many thesis topics and ideas around the “storage specialization” area of research.

We are in the process of acquiring the ZNS hardware (donated by Western Digital) and hopefully by the end of the course we can give you an opportunity to test your code on real hardware.

2. Logistics

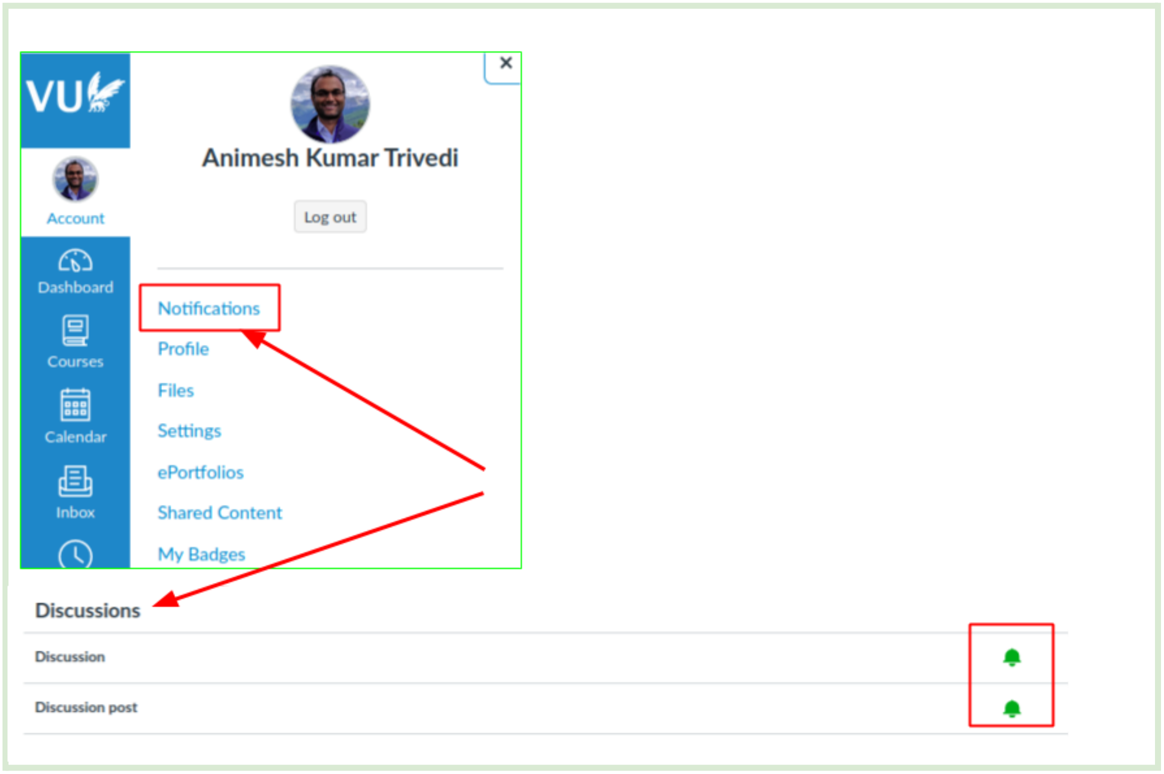
The project is the **team projec**t and consists of **5 milestones.** The first milestone is an individual milestone. This way you can assess your and your teammate’s capabilities before making teams during the first week. The next 4 milestones are group milestones, to be done in a group of **2 students**.

When are the deadlines: check the Canvas page to see when milestones are due.

Some pertinent and practical issues to consider:

- Be aware that you need to form a group in the first week. If you do not know anyone or have not managed to do so, then post it on the Discussion board of the Canvas page.
- You can choose to do the milestones alone, but we do not recommend that. You will not get any lenient marking from us just because you choose to do it alone.

Important: please enable notifications (https://canvas.vu.nl/profile/communication, you can also do course specific setting) for the Discussion forum as we will be posting many clarifications and information there. In any case do not forget to repeatedly check information there. See this image to enable notifications:



Weekly lab hours: There are weekly lab sessions. Please check the Canvas / Rooster. The lab hours are open to any sort of discussion including but not limited to questions about the project (we assume that would be the majority of the questions), clarification about lecture topics, general questions about research, etc.

2.1 What is given to you

We have provided a basic framework in which you will develop the project. This framework is a CMake project with example programs that should run at the end of each project milestone. Please make yourself familiar with CMake/C/C++ (and if needed gdb, and friends). We assume that you are familiar with the basic executable concepts in Linux (shared library, compilation, running and installing packages and programs from the source).

In the first milestone you are supposed to prepare a development environment, read the specification, and get yourself familiar with the project environment. This is a relatively simple milestone, and we would **strongly recommend** making use of this time to familiarize yourself with the concepts and source code, and the example programs.

Make sure to check out the due dates on Canvas for projects and ensure that you fulfill all the evaluation criterias. The whole framework and this complete handbook is released to you on day 1, and feel free to code ahead and finish as much as possible. On the demo day you can still just demo us the required functionality while being ahead in the project time. You can use the additional time for bonuses. See the bonus section later in this handbook.

Note: *we do not have any support to do this project on Windows or Mac or Windows' Linux subsystem. So please make sure that you have a working setup in the first week.*

2.2 How to ask for help when stuck in the project

Building any system is a complex task. The project that you build uses (or will use) many advanced concepts. So here is the protocol if you get stuck with issues

1. **Read the source code:** Everything given to you is a part of an open-source project and is open and accessible. Hence, the majority of issues or confusions could be clarified by just looking into the source code. We expect that you will spend time reading the framework given to you to understand its semantics. Such setup is very common in day-to-day life as a software developer or researcher where you need to quickly read and understand someone else's code.
2. **Ask Google:** Try to understand if the issue is related to the coding/Linux/C/C++/CMake/Bash environment and look for it appropriately. Often errors which might look strange can simply be resolved by an appropriate google search.
3. **Ask in the Discussion forum:** if the first two steps do not yield any results, then you can ask in the discussion forum. However, you should not ask for a direct solution to the milestones. Try to ask specific issues. We or other fellow students will jump in to answer your question. However, be prepared to hear that "debugging such issues is part of your coding milestones". Building any real system is a complex job, and you need to put in time to solve certain issues. There is no way around it.
4. **Ask in the lab hour:** If the issue is not time critical you can ask in the office hour and we can help you out there. You can share your screen and show us what went wrong with your code.
5. **Reach out to us:** If you believe that you are really stuck and there is something fundamentally wrong (hey, it can happen, no one is perfect ;)) please reach out to us. However, make sure to include what you have tried to do to solve the problem, what possibilities you have investigated and what you suspect is wrong. Try to include as much information as possible. Again: be prepared to hear ""debugging such issues is part of your coding milestones".

Important: We cannot guarantee any help hours before the deadline. So, please plan ahead, and ask with plenty of due time ahead. You only have one weekly lab session, try to make the most out of it.

2.3 Late/Slip days

In the course you have **3 slip days** that you can use to extend the deadline of code/quiz submissions (milestones 1, 2, and 4). There will be no extensions after that, and you will be awarded zero points.

Unfortunately you **cannot** use these days to move the interview dates. They are fixed, and you need to show whatever you have on that day.

2.4 Group Management Logistics

Please make sure that you regularly talk and coordinate with your group members and keep in touch. Meet early and plan ahead who does what and when to synchronize. If you are having troubles then contact us early enough. Too many times we have seen students showing up a day before the deadline saying that the team member decided not to work, or dropped out of the course - we cannot help a day before the deadline. We will consider exceptional circumstances like COVID related interrupts, however, beyond that there is very limited flexibility.

We will not be responsible for resolving disputes in your group. If you cannot do experiments as a group of 2, then do it alone, but you will not get a concession from us in grading for that.

3. Assessment Criteria

Each milestone is worth some points, and in total you can earn 50 points for the complete project. You can earn additional marks from bonuses, but you are restricted to **2 bonuses max** with a maximum of **additional 20 points (**on top of the maximum 50 points from the project). Hence, the total points achievable from the project work is 70.

Here is the evaluation plan for the milestones

- **Milestone 1** will be evaluated as a graded Canvas quiz, plus code upload
- **Milestone 2** is a code upload with the example target programming running successfully.
- **Milestone 3** will be evaluated with a group interview. All members of the group need to be presented in the interview session to answer questions.
- **Milestone 4** is a code upload with the example target programming running successfully.
- **Milestone 5** will be evaluated with a group interview. All members of the group need to be presented in the interview session to answer questions.

3.1 Grading guidelines

How the grading is done during the interview :

- We will first ask if your group have to project working
  - If yes, show us demo and explain what is happening
  - If not, then demonstrate to us to the best of your abilities (explain, code) where you are in the milestone.
- Explain various pieces of the code that you have written and what design and logic you have considered to implement and why
- The code will be judged on the generality (if you are making assumptions which are reasonable or not), error handling, hardcoding numbers, unexpected sleep/wait conditions to avoid race conditions, etc. We can not give a comprehensive list of criterias on which your code will be evaluated, however, we are reasonable people and we will explain what is being evaluated and why.
- Follow a coherent coding style, something like https://www.kernel.org/doc/html/latest/process/coding-style.html. Ugly code with unmanageable code segments will incur points penalties.

- You are graded on (1) if the code is working; (2) how well do you understand what is implemented - all team members, even if you split the work; (3) how well do you answer questions during the interview; and (4) how well do you understand the design space and choices present in your implementation of the code.

More interview logistics will be published as we approach the interview week.

Note on assessments and marking: Often students are under the impression that since they wrote a lot of code, and spent too much time in front of the computer they should get marks. Unfortunately this is not how programming assignments work. The project that is given to you has been done by us carefully and we know how much it should take as a team. This is a 6 ECTS course, and you need to put time and effort to get it done. A working code is the best proof of your time, energy, and effort put in this course.

Also - make sure to start early enough. I am sympathetic to the cause that this is _very_ coding heavy project. You have been warned. If you start one or two days before the deadline there is not much you will get done. The project is designed with a group effort worth 6 ECTS. Keep this in mind. I can not stress this enough: **Start early and make use of the lab session to ask questions and get feedback.**

3.2 Notes on partial assessment for Milestones

It might be the case that you do not have a working code for milestones 2, 3, 4 and 5 at the time of the interview or code upload. So here is the protocol for the assessment

1. You will first be interviewed on your general understanding of the milestone
2. If you demonstrated sufficient expertise then you can demonstrate your partial working code
3. In case you did not demonstrate sufficient understanding of the milestone, we will not review the code and maximum points awarded for this milestone would be whatever you achieve in the interview

*A non-working code on our machines will result in **maximum marks restricted to the 50% of the maximum marks possible**.*

4. StoSys Project Work

Background: NVMe is a command specification regarding how one should interact with NVMe devices which typically are high-performance non-volatile memory (NVM) storage devices such as Flash or Optane. The NVMe command set is designed to be lean, light-weight to simplify how many times system software needs to interact with the NVMe storage devices to complete an I/O request.

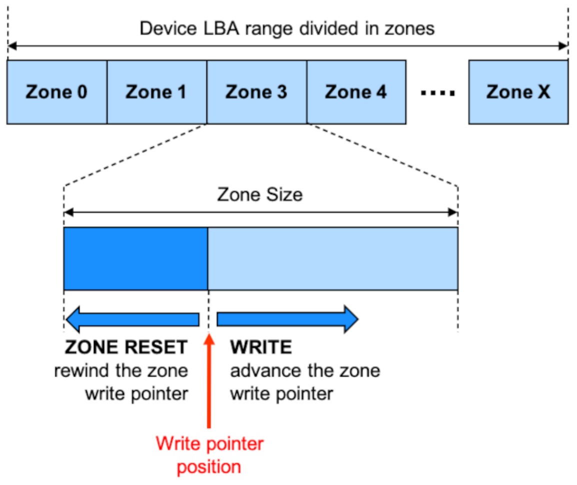

Device LBA range divided in zones
Zone 0
Zone 1
Zone 3
Zone 4
....
Zone X
Zone Size
ZONE RESET
rewind the zone write pointer
WRITE
advance the zone write pointer
Write pointer position

Figure 1: The principle of Zoned Device operations (taken from https://zonedstorage.io/introduction/zoned-storage/)

The idea of Zoned Devices, i.e. a device interface with append-only interface with segmented zones can be implemented with many device technologies including hard-disk drives (HDDs). With HDD it makes even more sense as HDDs have high penalty for random, rotational accesses. Hence, there are shingling HDDs in the market that you can buy today that follow the (almost) same Zone interface. Consequently, there are SCSI extensions for Zoned HDDs also (see https://www.t10.org/ftp/zbcr01.pdf). These specifications are implemented in a few userspace libraries such as libzbc, see https://zonedstorage.io/projects/libzbc/

Just like that, NVMe compliant devices also have Zone Complaint Specification (originally prepared as Technical Proposal TR 4053) which is now merged in the NVMe Specification 2.0 in August 2021 (*you are doing cutting edge research work).* The TP-4053 and NVMe 1.4b specification PDFs are given to you in the given framework (see the ./doc folder)[1].

In this project work we will use NVMe Zoned Namespace (ZNS) devices to do the practical work. ZNS devices are NVMe devices. They use NVMe read, write commands (among other management commands) together with controller and namespace identifiers. Apart from that, they have additional ZNS specific operations like ZNS controller and namespace identifiers, append, reset zone commands. All zone related efforts are very nicely documented at https://zonedstorage.io/

Here is a high-level development setup in which you will be doing the development during this project.

[1] *It might be confusing to follow all these details, but try to read through. We will only be using NVMe/ZNS devices in the project which are properly introduced.*

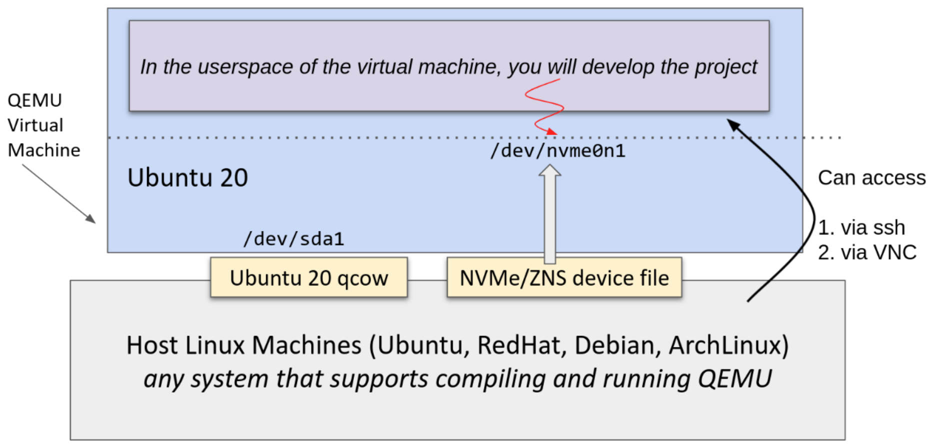

In the userspace of the virtual machine, you will develop the project
QEMU Virtual Machine
/dev/nvme0n1
Ubuntu 20
/dev/sda1
Can access
1. via ssh
2. via VNC
Ubuntu 20 qcow
NVMe/ZNS device file
Host Linux Machines (Ubuntu, RedHat, Debian, ArchLinux)
any system that supports compiling and running QEMU

Figure 2: Stosys project development setup with QEMU

A note on the terminology used:

- Block, Page, Sector - often (and sometimes confusingly) used to refer to the minimum I/O size for read and write operations. For example, HDDs have 512 bytes sectors. Flash devices can have 512 bytes of 4kB page size. Sometimes a page is also referred to as a block.
- Logical Block Address (LBA) - the address of the logical minimum read/write units.
- Zone address - the starting address of a zone. With ZNS the zones are clear. However, before ZNS sometimes the unit of garbage collection (which is the zone) was also referred to as a block. So just pay attention if we are talking about I/O operations or GC operations. Typically GC operates on larger and different units (like a couple of MegaBytes) than the basic I/O to amortize the cost of the GC operation.

When in doubt, just ask.

4.1 Milestone 1 (week 1) : *A new device is in town*

Type: individual
Maximum points: 10
What to do: prepare coding environment, code upload and take canvas quiz for milestone 1
Where: in the given framework, and canvas
Deadline: see the course canvas page

The goal of milestone 1 is to set up the development environment, and get familiar with the setup, framework, and ZNS and NVMe devices (and specification).

We are going to do the project development on top of Zoned Namespace (ZNS) devices. ZNS are NVMe devices and support all the basic NVMe commands (read, write, controller and namespace querying), plus ZNS specific details.

4.1.1 Task-1 setup QEMU 6.1

QEMU is a whole system emulator (with CPU, devices, BIOS, etc. everything) which is very popular to do real systems research with and is being deployed in the cloud. See https://www.qemu.org/docs/master/ for more details. We will do the project work in QEMU where the ZNS device will be emulated. The support for these devices was only added recently, hence, we need to compile from source QEMU.

Step 1: Download **QEMU 6.1** source code from https://download.qemu.org/qemu-6.1.0.tar.xz (or https://www.qemu.org/download/)

Step 2: Untar the code and then (you have to install some missing packages based on the Linux distribution that you are using as the host machine where QEMU is being compiled at)

```
$ tar xvf
$ mkdir build && cd ./build
$ ../configure --target-list=x86_64-softmmu --enable-kvm --enable-linux-aio
--enable-trace-backends=log --disable-werror --disable-gtk
$ make -j
```

Step 3: Once the compilation is done successfully, add the QEMU binary path in your .bashrc file (no need to install QEMU) as:

```
# find where the PATH variable is defined in your ~/.bashrc file
# put export PATH=<your_qemu_build_directory>:$PATH
# example of PATH line my .bashrc file:
export PATH=/home/atr/vu/qemu-6.1.0/build:$PATH

# do not forget to reload the new PATH variable
$ source ~/.bashrc
```

Step 4: At this point you should have QEMU build and ready to use, you can check it as (see the correct version number as 6.1.0)

```
$ atr@atr-xps-13:~$ qemu-system-x86_64 --version
QEMU emulator version 6.1.0
Copyright (c) 2003-2021 Fabrice Bellard and the QEMU Project developers
```

At this point we are ready to start the VM and install Ubuntu 20 in it.

4.1.2 Prepare the Ubuntu 20 in the QEMU VM

Pre-built Image: Maybe you do not want to see how sausages are made, and just want to get on with the project work. In that case, we have provided a cleanly pre-installed image of Ubuntu 20.04 LTS with 5.12 kernel at :
https://drive.google.com/drive/folders/19J-avjdbY0MQgPFzOWOdNC3ID1R5LN1w?usp=sharing

However, you still have to build QEMU 6.1 from source and make two NVMe and ZNS files as shown below. You can skip the Ubuntu 20.04 installation and v5.12 kernel building. For this image:

Username: user
Password: stosys!!

This user has sudo privileges. The base filesystem has 50GB. Feel free to modify this image and install whatever you like to facilitate your code development. I would not recommend a GUI and/or window-manager though. Learn how to do CLI-based code development and use git.

Just like a normal system QEMU needs a hard-disk image where the OS will be installed and booted from. We will also create a second image file that will be used as NVMe and ZNS devices (two different devices for testing and experiments).

Step 1 creating images: You can create a Ubuntu base hard-disk where the Ubuntu will be installed as :

```
# (RECOMMENDED) with qcow2 mode, does not consume 50G immediately
$ qemu-img create -f qcow2 ubuntu2004.img 50G

# otherwise, raw files will consume 50G immediately
$ qemu-img create -f raw ubuntu2004.img 50G
```

Now create two files which will be the NVMe and ZNS devices that we will experiment with:

```
# For ZNS and NVMe drives, we just create two raw files
$ qemu-img create -f raw znsssd-32M.img 32M
$ qemu-img create -f raw nvmessd-32M.img 32M
```

Step 2 installing the ubuntu 20.04 server system (no GUI): Download an ubuntu 20.4 server install iso from: https://releases.ubuntu.com/20.04/ubuntu-20.04.3-live-server-amd64.iso

Then start the QEMU VM as

```
$ sudo qemu-system-x86_64 -hda ubuntu2004.img --enable-kvm -m 4G -smp 2 -boot d
-cdrom ubuntu-20.04-live-server-amd64.iso -net user,hostfwd=tcp::7777-:22 -net nic
-cpu host
```

If with the start of the QEMU machine you get an error like:

```
Could not access KVM kernel module: No such file or directory
```

Try to probe the kvm module. If it is not found you might have to enable virtualization in the BIOS of your machine.

```
sudo modprobe kvm-intel
```

Once the system is running you can connect to the system using VNC using the output from the previous command run. You should see something like this in the terminal

```
VNC server running on 127.0.0.1:5900
```

You can connect to the virtual machine by putting this address in the VNC client. If your host system is also Ubuntu, you should have **Remmina** already installed (see Appendix A and B how to do remote desktop). You can paste the address "127.0.0.1:5900" in the address bar and change the protocol to VNC (not RDP) and connect (see Appendix B for more details). There is no special setup for the Ubuntu 20.04 QEMU installation. Follow the defaults to install the Ubuntu 20.04 image inside the QEMU virtual machine.

Remember to **enable SSH** during installation to be able to access the VM without VNC! At the end of your installation step, the system will ask if you want to install any new packages then pick **OpenSSH** server. If you missed this step, you can still login through VNC and then install the OpenSSH server. Once the installation is complete, you can stop the VM using Ctrl+C.

If you are having trouble installing the OS, see Appendix A for a detailed step by step guide.

Step 3 starting the system and ssh: Once the installation of the QEMU Ubuntu 20.04 is done, you can start the complete virtual machine with all the settings as

```
$ sudo qemu-system-x86_64 \
-name qemuzns -m 32G --enable-kvm -cpu host -smp 4 \
-hda ubuntu-2004.qcow \
-net user,hostfwd=tcp::7777-:22 -net nic \
-drive file=znsssd-32M.img,id=zns-device,format=raw,if=none \
-drive file=nvmessd-32M.img,id=nvme-device,format=raw,if=none \
-device nvme,serial=zns-dev,id=nvme2,zoned.zasl=5 \
-device
nvme,drive=nvme-device,serial=nvme-dev,physical_block_size=4096,logical_block_size=
4096 \
-device
nvme-ns,drive=zns-device,bus=nvme2,nsid=1,logical_block_size=4096,physical_block_si
ze=4096,zoned=true,zoned.zone_size=1M,zoned.zone_capacity=1M,zoned.max_open=0,zoned
.max_active=0,uuid=5e40ec5f-eeb6-4317-bc5e-c919796a5f79
```

Make sure to use the right file location for the files which are highlighted in yellow above. We are not going into the details of what each of these QEMU parameters do. Please see here :

- QEMU documentation https://www.qemu.org/docs/master/system/index.html.
- NVMe documentation https://www.qemu.org/docs/master/system/devices/nvme.html
- Further information: https://zonedstorage.io/getting-started/zns-emulation/

After a few seconds you should be able to ssh into the VM as (see the QEMU command above, hostfwd, which forwards traffic from local 7777 port to VM's 22 port which is the port used by the ssh server)

```
$ ssh -X user@localhost -p 7777
```

Otherwise use the username and password you set up during the Ubuntu 20.04 installation. For any subsequent ssh instruction we will use the provided one.

Step 4 upgrading the Linux kernel: Since NVMe and ZNS development is very cutting edge, the base kernel that comes with Ubuntu 20.04 (v5.4) is too old. See this: https://zonedstorage.io/linux/overview/ (only v5.9 and above has support for ZNS devices).

Now we are going to upgrade the kernel. You can choose the latest kernel version, but I strongly recommend to use the **5.12 version** - this is what I tested extensively. The following compiling instructions are based on: https://wiki.ubuntu.com/KernelTeam/GitKernelBuild. To compile the v5.12 kernel (you do not have to do this inside the VM, just find the fastest Linux machine you can which can compile a kernel. Any distro will do, but x86 architecture is recommended unless you know how to cross compile):

```
$ git clone https://git.kernel.org/
$ cd kernel (or whatever folder it made)
$ git checkout v5.12

# make sure all packages are installed to build a kernel
$ make olddefconfig
# or you can also do
$ make menuconfig

# now we build
$ make clean
$ make -j $(getconf _NPROCESSORS_ONLN) deb-pkg LOCALVERSION=-stosys
```

After a successful make (can take 20-30mins), you should have a bunch of deb files in your parent folder where you compiled the kernel. Then copy the final deb files inside the VM and install them. I did this like this (make sure you change the file names to what you have):

```
# on the machine you compiled the kernel (make sure that the VM is running)
$ scp -P 7777 linux-image-5.12.0+_5.12.0-stosys-6_amd64.deb user@localhost:~/
$ scp -P 7777 linux-headers-5.12.0+_5.12.0-stosys-6_amd64.deb user@localhost:~/

# inside the VM
$ sudo dpkg -i linux-image-5.12.0+_5.12.0-stosys-6_amd64.deb
$ sudo dpkg -i linux-headers-5.12.0+_5.12.0-stosys-6_amd64.deb
$ sudo upgrade-grub2
$ sudo sync
$ sudo reboot
```

At this point, your VM should boot into the 5.12 kernel. You can skip all these steps and can just use the provided 5.12 image directly and move to the next subsection.

4.1.3 Task-3 Familiarize yourself with online resources

The following are the key resources that you should read through to get a conceptual understanding of ZNS devices and the Linux development environment and integration.

1. In general, I would strongly recommend to flip through the whole https://zonedstorage.io website. You may find yourself looking for details here over and over again.
    a. Understand what zoned storage is (broader concept): https://zonedstorage.io/introduction/
    b. Introduction to NVMe ZNS concepts : https://zonedstorage.io/introduction/zns/
    c. libnvme user library https://zonedstorage.io/projects/libnvme/ (you will be using this library in the project)
    d. Understand the broad Linux and ZNS ecosystem: https://zonedstorage.io/linux/
2. You ***should*** flip through the **NVMe 1.4 specification and ZNS TR 4053** (given in the framework, see the ./docs/ folder).
    a. In the NVMe 1.4 specification you should read through section: **chapter 1**, sections **3.1** (has details about controllers capabilities and metadata), section **4.1, 4.2, 4.6** (explains the basic I/O data structures. **4.6.1.2** contain error codes), section **5.15** (identify namespace **5.15.2.1** and the NVMe controller **5.15.2.2** are important), **6.15** (write command), and **6.9** (read command). Many of the details of these will be hidden behind the library that you would use but you still need to fill out function parameters which are defined and explained in these sections.
    b. In the TP 4053 you should read: chapter **2**, section **3.1** (ZNS namespace and controller identification structures, section **4.1** (ZNS errors), **4.3** (how to interact with zones, including resetting and closing them), **4.4** (explains how interact with zones and extract their current state), **4.5** (special zone append command). Notice figure 20 for an overview.

    The idea of reading these specifications is not to memorize them, but to make sure you get an intuition how the information is organized, and how you can quickly look for specific commands or details. Also, you do not have to read them all at once, instead read them gradually as you move with the coding.

4.1.4 Task-4 Build and run nvme cli command tool

Inside the Ubuntu 20.04 (with 5.12 kernel) VM we are now going to interact with NVMe and ZNS devices. The primary way a user can interact with an NVMe device is via using nvme command line interface tools (nvme-cli). This command/package is available on most of the popular Linux distributions. However, since we need support for the latest ZNS devices, we need to clone and build from source.

nvme-cli https://github.com/linux-nvme/nvme-cli

I have tested the code with v1.14. Use the specific commit that we show below. See the README.md file in the repo on how to compile the code.

```
#install the dependency
$ sudo apt install uuid-dev

git clone https://github.com/linux-nvme/nvme-cli
$ cd nvme-cli
$ git checkout f38d0c1c20db1c8e870ba828be1bd960511dba98
$ git submodule init
$ git submodule update
$ make -j
…

$ sudo ./nvme -version
nvme version 1.14.121.gf38d
$ sudo ./nvme list
Node             SN           Model                    Namespace Usage               Format      FW Rev
---------- ------------ -------------------- ------ ------------------------ -------------- --------
nvme1n1      nvme-dev    QEMU NVMe Ctrl       1     8.39  MB /   8.39  MB    4 KiB +  0 B   1.0
nvme0n1      zns-dev     QEMU NVMe Ctrl       1     8.39  MB /   8.39  MB    4 KiB +  0 B   1.0
```

At this point we are ready to interact with the NVMe devices using the command line interface using our newly compiled nvme command. In the example above you can see the two NVMe devices (one normal and one ZNS is shown). *I used 8MB devices for testing, hence 8MB capacity*. You can also put the location of the nvme command binary in your $PATH variable in bashrc if you do not want to always run the command from the location.

For more examples of how you can use nvme cli tool to interact with the ZNS and/or NVMe devices see: https://zonedstorage.io/projects/zns/

Important: the way the setup is done, /dev/nvme0n1 is your ZNS device and /dev/nvme1n1 is your ordinary NVMe device.

Your particular tasks is to run these command, fill in the details (*find any valid value to fill out the yellow parts and see if you understand it*), understand their output and match that with the specification (make yourself familiar with how to read the NVMe and ZNS specifications):

1. Extract and identify NVMe controller and Namespace structures for the NVMe device and match the details from the NVMe specifications (5.15.2.1 and 5.15.2.2). Hint use -H flag and compare the output with specification.

```
# read the help in the commands
$ nvme help
$ nvme id-ctrl help
$ nvme id-ns help

# now fill in the command
$ sudo nvme id-ctrl [dev] -H (fill the yellow part)
$ sudo nvme id-ns [dev] -H (fill the yellow part)
```

2. Extract and identify ZNS controller and Namespace structures for the ZNS device and match them with the ZNS specification (3.1.1 and 3.1.2)

```
$ nvme zns help
$ nvme zns id-ctrl help
$ nvme zns id-ns help

# now fill in the command
$ sudo nvme zns id-ctrl [dev] -H (fill the yellow part)
$ sudo nvme zns id-ns [dev] -H (fill the yellow part)
```

3. Execute NVMe LBA read and write commands and test them using -d flag. With -d (or --data) you can write and read data from files, and then compare their checksums. Keep in mind the ZNS device uses the same NVMe read and write commands.

```
# read the help in the commands
$ nvme read help
$ nvme write help

# now fill in the command to check if you can read, write and compare
outputs
# make a random 4kb file
$ dd if=/dev/random of=./4kb bs=4096 count=1
# now we write and read this file, and compare the content

$ sudo nvme write [dev] -s _ -z _ -d ./4kb (fill the yellow part)
$ sudo nvme read [dev] -s _ -z _ -d ./4kb-w (fill the yellow part)

#compare the data, the checksum should match
$ md5sum ./4kb
075ca305497df6cfb9500abb98993438  ./4kb
$ md5sum ./4kb-w
075ca305497df6cfb9500abb98993438  ./4kb-w

# now let's try to rewrite the ZNS again with the same command and same values!
$ sudo nvme write [dev] -s _ -z _ -d ./4kb (fill the yellow part)
??? ← What output do you get? Do you get the same output with NVMe?
```

4. Execute ZNS zone report and Zone reset commands:

```
# read the help in the commands
$ nvme zns report-zones help
$ sudo nvme zns report-zones [dev] -s ___ -H (fill the yellow part)
```

```
# reset zones
$ sudo nvme zns reset-zone [dev] -s  ___ (also see -a)
```

5. Execute ZNS zone append command

```
# read the help in the commands
$ sudo nvme zone-append help

# fill in the command:
$ sudo nvme zone-append [dev] -s _ -z _ -d ./4kb

# now run this command 3 times,
$ sudo nvme zone-append [dev] -s _ -z _ -d ./4kb
$ sudo nvme zone-append [dev] -s _ -z _ -d ./4kb
$ sudo nvme zone-append [dev] -s _ -z _ -d ./4kb

# Where is the zone write pointer?
$ sudo nvme zns report-zones  [dev]
```

Why are we doing so many practice commands? When you run into issues with your project code, you can compare the behavior of the ZNS/NVMe devices with these commands (with similar I/O pattern) and see if you are making any mistake the way you are interpreting commands, their results in your code. You can dump the state of the ZNS device (zone state, write pointers) and diagnose problems. Plus, how these commands are implemented is giving you a reference code to browse. Feel free to read the nvme cli code to get a better understanding.

4.1.4 Task-4 Build and prepare the coding environment

The programmatic way to interact with the NVMe/ZNS devices is through libnvme userspace shared library (among other ways, see bonuses for that). In this task we are going to compile the libnvme library from source (with some patches from me) to make sure things work. At the end of this step you should be able to compile the given framework to you for milestone 1.

Copy the given framework with in the virtual machine

```
# from the host machine copy it inside the VM
$ scp -P 7777 msc-stosys-framework-skeleton.zip user@localhost:~/

# ssh into the VM
$ ssh -X user@localhost -p 7777

# unzip and setup
$ sudo apt install unzip (if needed)

$unzip msc-stosys-framework-skeleton.zip
```

The patched version of the library is given to you in the framework (the actual library is located at https://github.com/linux-nvme/libnvme). The patch is nothing special, except that libnvme did not export ZNS specific functions. More recently they have moved their build to meson, which I have not tested. Furthermore, when I prepared the project, the libnvme scan was broken in Linux, so I had to work around it. In order to avoid any unforeseeable issues with the library (a fast moving coding repo). I have a fork that is given to you in the framework. Here is how you can compile and install it:

```
# install the package config dependency
$ sudo apt install pkg-config

# then build and compile
$ cd msc-stosys-framework/libnvme
$ ./configure
$ make -j
$ sudo make install

# if you want to install this in a non-standard path, you can
$ ./configure --prefix=/home/$USER/local/
[...] ← see the output here with the appropriate path names
$ make -j
# if you install in some non-standard path, then you probably do not need sudo
$ make install

# make sure to put your non-standard install (if you did that) path in your -I
(gcc) and LD_LIBRARY_PATH
```

Once you have installed the library, you can prepare the framework:

```
$ cd msc-stosys-framework
$ cmake .
-- Building using CMake version: 3.16.3
compiler is GNU   and name is g++
[info] compiler is gcc/g++
[info] ASAN is on, adding additional ASAN fields to the compiler
-- Configuring done
-- Generating done
-- Build files have been written to: /home/atr/msc-stosys-framework
$ make -j
```

At this point you are ready to finish the code for Milestone 1.

4.1.5 Task-5 Finish the missing code pieces and take canvas quiz

Milestone 1 (in m1.cpp) does the following things:

1. Does a full system scan to find all devices (similar to the `nvme list` command as shown above), return a count of all the devices.

2. Does a second scan, but this time splits the found NVMe devices into ZNS-capable or not devices, and returns an array of `struct ss_nvme_ns` devices (size of the array is the number of devices found).
3. Pick the last ZNS capable device and enumerate its properties (`show_zns_zone_status`)
4. Then it performs two I/O tests:
    a. Test 1 `test1_lba_io_test:` In this function, a series of LBA size tests are conducted. This includes: read, write, zone reset, and zone append.
    b. Test 2 test2_zone0_full_io_test: In this function, a full zone size I/O test (reset, write, and read) are tested.
5. Close everything and finish.

Your goal is to fill up functions in the device.cpp so that the aforementioned tests pass. More specifically, you need to provide functions for (you are free to change the function signature to pass any required details):

- ss_nvme_device_read(...): Write a function to read one or more LBA addresses in a user provided buffer from an NVMe device (the same can be used for ZNS devices).
- ss_nvme_device_write(...): Write a function to write one or more LBA addresses from an NVMe device (the same can be used for ZNS devices).
- ss_zns_device_zone_reset(...): Write a function to reset a given zone on the ZNS device.
- ss_zns_device_zone_append(...): Write a function to append data in a given zone on the ZNS device. Keep in mind, appending also has a maximum append size (ZASL, see Figure 10 in section 3.1.2 in the ZNS TR 4053).
- update_lba(...): Update LBA function takes an LBA address and returns the next LBA address that can be used to write data with a certain offset. For example, if you start writing from LBA 0 with a data size of 2 x LBA size (2 LBA blocks), then the function should return the next LBA address to write after writing the 2x LBA data. It is important to understand this concept because all zones have an associated write pointer in them that is internally incremented with each write in zones, and you must understand how this is done. If you try to write an LBA in the zone which does not match the zone-internal write pointer then you will get an error.
- get_mdts_size(...): Return the maximum data transfer size in bytes. See the QEMU MDTS bug below.
- ss_nvme_device_io_with_mdts(...): Write a wrapper function around read/write functions (defined above) where a large I/O request can be split into multiple MDTS size I/O requests. You can pass MDTS size as an argument in the function. For example, if an NVMe device has MDTS size of 512KB, then a complete zone read/write of 1MB should be split into 2x 512 KB I/O requests.

Almost all of these functions can be implemented in *10s of lines of code* by directly calling appropropriate functions in libnvme. Browse libnvme code and header files.

Calculating MDTS is challenging. MDTS is defined in the identify controller data structure (5.15.2.2 Identify Controller data structure (CNS 01h)), but the unit is (CAP.MPSMIN) page size. The value MPSMIN (indicates the minimum host memory page size that the controller supports, see section 3.1.1 Figure 69: Offset 0h: CAP – Controller Capabilities) is defined in a Read-Only Capability register (as PCIe BAR). You need to figure out how to read this register (**Hint:** see `nvme show-regs` command, and corresponding code in the nvme-cli repository). See Issue-2 below with the QEMU MDTS bug.

Please pay attention to compiler warnings and memory leak errors. We will look at them properly and deduct marks, if appropriate. See the CMakeLists to see which flags are we using.

Example of a successful run:

```
$ sudo ./bin/m1
==============================================================
Welcome to M1. This is lot of ZNS/NVMe exploration
==============================================================
nvme-subsys1 - NQN=nqn.2019-08.org.qemu:nvme-dev
 `- nvme1 pcie 0000:00:05.0 live
   `- nvme1n1lba size:4096 lba max:2048
nvme-subsys0 - NQN=nqn.2019-08.org.qemu:zns-dev
 `- nvme0 pcie 0000:00:04.0 live
   `- nvme0n1lba size:4096 lba max:2048

total number of devices in the system is 2
root (0) |- name: nvme-subsys1 sysfs_dir /sys/class/nvme-subsystem/nvme-subsys1
subsysqn nqn.2019-08.org.qemu:nvme-dev
       |- controller : name nvme1 (more to follow)
              |- namespace : name nvme1n1 and command set identifier (csi) is 0 (= 0
NVMe, 2 = ZNS), more to follow)
 root (1) |- name: nvme-subsys0 sysfs_dir /sys/class/nvme-subsystem/nvme-subsys0
subsysqn nqn.2019-08.org.qemu:zns-dev
       |- controller : name nvme0 (more to follow)
              |- namespace : name nvme0n1 and command set identifier (csi) is 2 (= 0
NVMe, 2 = ZNS), more to follow)
namespace: nvme1n1 and zns NO
namespace: nvme0n1 and zns YES
Opening the device at nvme0n1

....

zone is APPENDED 2x successfully, returned pointer is at 102 (to match 102)
The final write is ok too, we should be at 5x LBAs writes now
The final 5x read is ok, matching pattern ...
        testing the 0 buffer out of 5... passed
        testing the 1 buffer out of 5... passed
        testing the 2 buffer out of 5... passed
        testing the 3 buffer out of 5... passed
        testing the 4 buffer out of 5... passed
ZNS I/O testing OK
Test 3: testing the max writing capacity of the device, trying to read and write a
complete zone of size 1048576 bytes
        trying to reset the zone at 0x100
        the whole zone reading done
OK: the whole zone pattern matched
```

```
====================================================================
Milestone 1 results
Test 1 (read, write, append, reset) :  Passed
Test 2 (Large zone read, write)     :  Passed
====================================================================

=================================================================
==4361==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 118 byte(s) in 4 object(s) allocated from:
    #0 0x7fe42d6f43dd in strdup (/lib/x86_64-linux-gnu/libasan.so.5+0x963dd)
    #1 0x7fe42d64fba6 in nvme_create_ctrl nvme/tree.c:898

Direct leak of 20 byte(s) in 4 object(s) allocated from:
    #0 0x7fe42d6f43dd in strdup (/lib/x86_64-linux-gnu/libasan.so.5+0x963dd)
    #1 0x7fe42d64db18 in __nvme_get_attr nvme/util.c:803

SUMMARY: AddressSanitizer: 138 byte(s) leaked in 8 allocation(s).
```

After finishing - upload the code and take the canvas quiz.

Issue 1: libnvme has memory leaks (*this _might_ be fixed in the master, but I have not verified it yet*)

```
=================================================================
==49407==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 118 byte(s) in 4 object(s) allocated from:
    #0 0x7f47010883dd in strdup (/lib/x86_64-linux-gnu/libasan.so.5+0x963dd)
    #1 0x7f4700fe3ba6 in nvme_create_ctrl nvme/tree.c:898

Direct leak of 20 byte(s) in 4 object(s) allocated from:
    #0 0x7f47010883dd in strdup (/lib/x86_64-linux-gnu/libasan.so.5+0x963dd)
    #1 0x7f4700fe1b18 in __nvme_get_attr nvme/util.c:803

SUMMARY: AddressSanitizer: 138 byte(s) leaked in 8 allocation(s).
```

Issue 2: QEMU MDTS bug

There is some issue (unofficially tested, but not confirmed) that whatever MDTS QEMU NVMe reports, use one less than that. For example, if the MDTS reported is 6, then use 5. In my testing the NVMe (and ZNS) device fails non-deterministically to complete the I/O with the reported MDTS size. However, this does not happen on real NVMe hardware (like NVMe Optane or Flash), only in QEMU NVMe devices.

4.1.6 All together

Specific goals for M1: To recap this milestone (as there are many small things to do):

1. Make sure you have a working development setup with QEMU and ZNS
2. Read the online documentation (https://zonedstorage.io/), TR 4053, and NVMe 1.4b specifications (only the sections prescribed above. The whole 1.4b specification is 400+ pages).
3. Make sure nvme-cli is working and you can browse the code

4. Complete the Milestone 1 missing functions and make sure the milestone 1 tests passes
5. Upload the code on Canvas as a zip file with the format : milestone1.zip (or tar or whatever) Check the upload deadline on canvas.
6. Take the canvas quiz
7. [**Not graded**] Find yourself a team-mate to work as a team for Milestone 2 onwards. When choosing a teammate ask them about the details of this milestone to get a good skill match.

Important: *In the quiz we will ask you to run certain commands and find the right answer from the options given. So it is very important that you keep the VM running when you take the quiz.*

4.2 Milestone 2 (week 2) : I can't read, is there a translator here?

Type: group project
Maximum points: 10
What to do: implement a hybrid log-structured FTL to hide ZNS device complexity (*without GC*) and upload the code
Where: in the framework (code upload on canvas)
Deadline: see the course canvas page

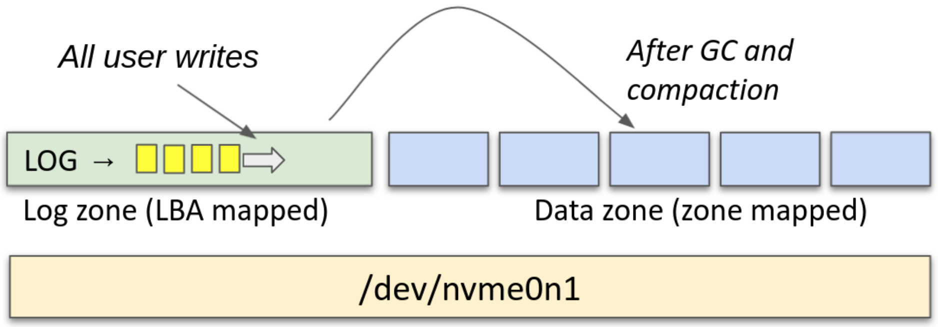


Figure 2 : A high-level layout of the hybrid log-structured FTL design

The purpose of this milestone is to design and implement a **hybrid log-structured flash translation layer (FTL)** on top of the ZNS device. With this FTL in place, the ZNS device should appear as a normal block device where a user can read and write any block offset without any restriction.

The design of the hybrid log-structured FTL is discussed in the class. See the class slides.

The key design decisions that you have to make is
1. What fraction of the device zones you should use for the write log, and what fraction for the data zone
2. How to manage translation mappings in the log and data segments (use a zone-level or page-level translation lookups)
3. [Plan] what will GC do and when a log content will be moves to a Data zone, and vice-versa

When designing and implementing the FTL, make sure you understand how ZNS devices are expected to be read, written and reset.

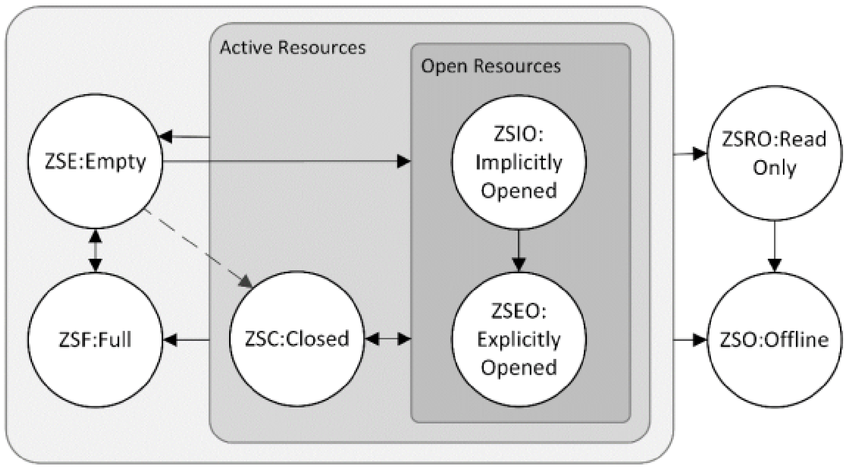


Figure 3: NVMe ZNS zone state machine

Practical information: Milestone 2 (M2) is decoupled from milestone 1. M2 is built as a shared library (`libstosys.so`) which is then eventually linked against the m2.cpp code (the main file which runs the tests). You should aim to implement high-level functions (your FTL device open, read, write, and close) as given in the `zns_device.h` file. In that file there is also a `struct user_zns_device.` This structure is the primary device structure that should be used everywhere in your code. There are following fields:

1. `uint32_t lba_size_bytes:` Logical block size that the user should adhere to when doing I/O in this device. This can be the same as the underlying LBA size of the NVMe/ZNS device or some multiple of it.
2. `uint64_t capacity_bytes:` Total user-usable capacity in bytes. This would be the actual ZNS device capacity, minus the FTL metadata and log zones. A user should be able to read/write this area [0, capacity_bytes] in lba_size_bytes pages any number of times (for this milestone without GC, only once).
3. `void *_private:` A private pointer area that you can use to save any information (like your own private structure with information). The user is not supposed to touch or change this.

You need to implement the following functions:

1. `int init_ss_zns_device(char *name, struct user_zns_device **my_dev)`: this function creates a user-device with appropriate values.
2. `int zns_udevice_read(struct user_zns_device *my_dev, uint64_t address, void *buffer, uint32_t size)`: a user-facing read function. A user should be able to read any arbitrary LBA address (any size, which is multiple of the block size).
3. `int zns_udevice_write(struct user_zns_device *my_dev, uint64_t address, void *buffer, uint32_t size)`:a user-facing write function. A user should be able to write to any arbitrary LBA address (any size, which is multiple of the block size).
4. `int deinit_ss_zns_device(struct user_zns_device *my_dev)`: a shutdown close function for a device.

The m2.cpp file has following tests that your FTL implementation should pass (at this stage)

1. Test 1 - a simple LBA zero write and read (and pattern matching)

2. Test 2 - a 32 block sequential writing at a randomly chosen starting LBA (and pattern matching). 32 is just chosen randomly assuming that it will not trigger any GC issues, otherwise decrease it.

As you may have noticed the mix of C and C++ languages in the project. Feel free to do so, but be careful with the linkage between C and C++ (use appropriate extern "C" { } blocks)

Plan ahead: You do not have to implement GC and support for overwriting blocks as in the current testing there are none. This will come in milestone 3 with the garbage collection support. But we advise to plan for it already in the choices of data structures you have. Also, persistence (shutdown and resume) in the milestone 5 (so plan a bit ahead).

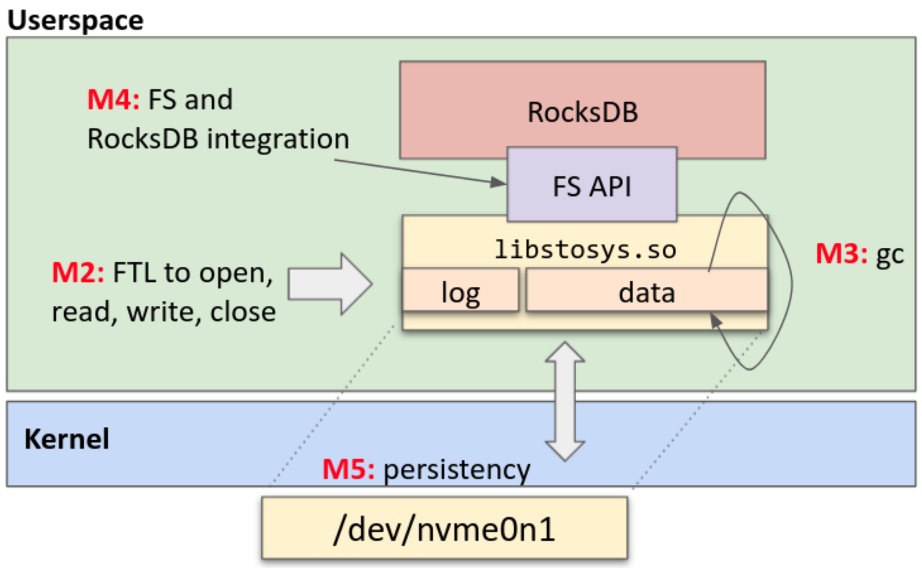


Figure 4: Various milestones and their location in the overall architecture.

Specific goals for M2:

1. Design and implement a hybrid log-structured FTL which
   - Gives user a block device to read and write at any LBA address (captured in `struct user_zns_device)`
   - Keeps track of log and data mappings
2. Make sure the m2 tests pass (execute ./bin/m2 /dev/nvme0n1)
3. Upload the code on Canvas as a zip file with the format : milestone2-groupNN.zip (or tar or whatever). Check the upload deadline on canvas.

Successful example run for the m2 milestone

```
$ make -j
# pass the ZNS device
$ sudo ./bin/m2 /dev/nvme0n1
=====================================================================
This is M2. The goal of this milestone is to implement a hybrid log-structure ZTL
(Zone Translation Layer) on top of the ZNS (no GC)
=====================================================================
Opening the namespace device: nvme0n1
[...]
4096 bytes written successfully on lba 0x0
4096 bytes read successfully on lba 0x0
Writing of 32 unique LBAs OK
Reading and matching of 32 unique LBAs OK
====================================================================
Milestone 2 results
Test 1 (write, read, and match on LBA0)   :  Passed
Test 2 (32  LBA write, read, match)       :  Passed
====================================================================
```

4.3 Milestone 3 (week 3) : It's 2021, we recycle

Type: group project
Maximum points: 10
What to do: implement the GC logic in the FTL; (ii) upload the code; and (iii) show up for the interview
Where: code upload on canvas, and on the campus interview
Deadline: see the course canvas page

In the previous milestone, we wrote the LBA address 0 and a random offset of some LBAs. We try to make sure we do not write the same address again or overflow the log zone. In this milestone you will build a GC capable FTL. With the GC, the device will become a completely general purpose read/write-able block device. The GC logic is completely FTL's internal business and no API changes are necessary to the `zns_device.h` file that is developed in the Milestone 2.

When does a GC occur? (1) when you run out of the log blocks; (2) during active wear leveling. In this Milestone you can focus on the case (1) only.

GC Design choices: Consider what GC algorithm you want to implement, how do you want to ensure ordering, what kind of information you want to track about hot-cold pages or blocks, what is the GC kick-in threshold, how do you select a victim block from the log, how do you move data around, etc. (Hint: maybe not all these need to be answered for a hybrid log FTL design).

Important: You are NOT allowed to implement an in-place GC, where GC only erases a block/zone whenever a new write comes in the same user-thread. This way a new write could result in an erase-merge-write cycle. The GC MUST be implemented as a separate thread in your FTL, and run independently from the rest of the user-driven I/O requests. Of course, that means that the GC thread must take the right locks and synchronize with other threads not to erase log zones when they are being concurrently written or read. This overall thread-safe architecture will keep in handy in Milestone 4 and 5 as well.

Testing: The testing framework is similar to milestone 2. Specifically, the tests are given in the `m3.cpp` file. Please read through the file to understand the intent of the tests. In the file, we have following tests:

1. Write, read, and match pattern over the whole device sequentially once (from LBA 0 to max LBA)
2. Write, read, and match pattern over the whole device randomly once
3. Write, read, and match pattern over the whole device by randomly hammering certain LBA locations for "x" number of times

While writing the data to the ZNS device, the test also writes data in parallel to a local shadow file and then compares the results from what was read from the device and what is in the file. The file is deleted at the end of the test. If you want to hold on to the file, feel free to modify the test program and remove the remove_file call.

Feel free to add your own tests.

Specific goals for M3:

1. Enhance your FTL design with a GC, with the GC
    a. Zns device can be read from or written to any number of times
2. Make sure the m3 tests pass (execute `./bin/m3 /dev/nvme0n1`)
3. Upload the code on Canvas as a zip file with the format : milestone3-groupNN.zip (or tar or whatever) Check the upload deadline on canvas.
4. **Show up for the interview**, give demo, and explain you code

Example successful execution of m3

```
$ make -j
[...]
$ sudo ./bin/m3 /dev/nvme0n1
====================================================================
This is M3. The goal of this milestone is to implement a hybrid log-structure ZTL
(Zone Translation Layer) on top of the ZNS WITH a GC

^^^^^^^^^
======================================================================
Opening the namespace device: nvme0n1
device nvme0n1 opened successfully 3
[...]
fallocate OK with ./tmp-output-fulld and size 0x500000
the ZNS user device has been written (ONCE) completely OK
verifying the content of the ZNS device ....
Verification passed on the while device
fallocate OK with ./tmp-output-fulld and size 0x500000
the ZNS user device has been written (ONCE) completely OK
verifying the content of the ZNS device ....
Verification passed on the while device
fallocate OK with ./tmp-output-fulld and size 0x500000
the ZNS user device has been written (ONCE) completely OK
Hammering some random LBAs 10000 times
Hammering done, OK for 10000 times
verifying the content of the ZNS device ....
Verification passed on the while device
====================================================================
Milestone 3 results
Test 1 sequential write, read, and match (full device)                :  Passed
Test 2 randomized write, read, and match (full device)                :  Passed
Test 3 randomized write, read, and match (full device, hammer 10000 ) :  Passed
```

```
===================================================================
```

4.4 Milestone 4 (week 5) : We love Rock(sDB) 'n' Roll!

Type: group project
Maximum points: 10
What to do: integrate your FTL with a file system design into RocksDB key-value store, and code upload
Where: in the framework, and code upload on Canvas
Deadline: see the course canvas page

IMPORTANT: Since RocksDB FS API is in C++, use C++ as the choice of language. Be aware, *use the **CC** file extension, not **CPP** here*. I need to look through it systematically but I had issues with cpp file extension not compiling inside RocksDB. To be on the safer side, just use the extension cc for this milestone.

At this point, you (should) have a working FTL implementation with GC. It is time to put it to a good use and integrate into a widely used key-value store called RocksDB. RocksDB is an embedded key-value store, i.e., it provides an efficient KV interface for users to store data locally. RocksDB uses LSM trees to store and read key-value data. More about it can be found at https://github.com/facebook/rocksdb/wiki

It is a very large project with many advanced topics. Naturally, we cannot cover all these details in a short 5 week class project. In this milestone, we are going to focus on writing a file system engine for it on top of the ZNS device with FTL that you developed in the previous milestone.

Step 1: Get and compile RocksDB with a file system hook for your filesystem

Install all RocksDB dependencies:
https://github.com/facebook/rocksdb/blob/main/INSTALL.md#dependencies

```
$ sudo apt install autoconf libgflags-dev libtool autoconf-archive
$ sudo apt install libsnappy-dev zlib1g-dev libbz2-dev liblz4-dev libzstd-dev

$ git clone https://github.com/westerndigitalcorporation/libzbd.git
$ cd libzbd

# I tested the code with this
$ git checkout e9c593685e1c957bc903d3b937dffc7f15710324

$ sudo apt install autoconf automake libtool m4 autoconf-archive
$ sh ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

Then we need to get RocksDB and make it aware of our FTL/file system code.

```
$ git clone https://github.com/facebook/rocksdb.git
$ cd rocksdb

# may be the master would also work, but I tested the code with this
$ git checkout c0ec58ecb94d276f69ccdfa956c5152d4c91a2ec

# now we need to copy the folder content from the framework into rocksdb plugin
folder (update the paths accordingly what you have on your machine)
$ mkdir plugin/s2fs
$ cp -r  msc-stosys-framework/src/rocksdb-plugin/* rocksdb/plugin/s2fs/

# now we compile (below is one line command)
$ DEBUG_LEVEL=0 ROCKSDB_PLUGINS="s2fs" USE_RTTI=1 PREFIX=/home/$USER/local/
DISABLE_JEMALLOC=1 DISABLE_WARNING_AS_ERROR=1 make -j4 db_bench install
…
[at the end]
  CC       utilities/transactions/lock/range/range_tree/range_tree_lock_tracker.o
echo 'prefix=/home/atr/local/' > rocksdb.pc
echo 'exec_prefix=${prefix}' >> rocksdb.pc
echo 'includedir=${prefix}/include' >> rocksdb.pc
echo 'libdir=/home/atr/local//lib' >> rocksdb.pc
echo '' >> rocksdb.pc
echo 'Name: rocksdb' >> rocksdb.pc
echo 'Description: An embeddable persistent key-value store for fast storage' >>
rocksdb.pc
echo Version: 6.25.0 >> rocksdb.pc
echo 'Libs: -L${libdir} -ldl -lrocksdb' >> rocksdb.pc
echo 'Libs.private: -lpthread -lrt -ldl -lsnappy -lgflags -lz -lbz2 -llz4 -lzstd
-lnuma  -u stosys_s2fs_reg' >> rocksdb.pc
echo 'Cflags: -I${includedir} -std=c++11  -faligned-new -DHAVE_ALIGNED_NEW
-DROCKSDB_PLATFORM_POSIX -DROCKSDB_LIB_IO_POSIX  -DOS_LINUX -fno-builtin-memcmp
-DROCKSDB_FALLOCATE_PRESENT -DSNAPPY -DGFLAGS=1 -DZLIB -DBZIP2 -DLZ4 -DZSTD -DNUMA
-DROCKSDB_MALLOC_USABLE_SIZE -DROCKSDB_PTHREAD_ADAPTIVE_MUTEX -DROCKSDB_BACKTRACE
-DROCKSDB_RANGESYNC_PRESENT -DROCKSDB_SCHED_GETCPU_PRESENT
-DROCKSDB_AUXV_GETAUXVAL_PRESENT -march=native   -DHAVE_SSE42  -DHAVE_PCLMUL
-DHAVE_AVX2  -DHAVE_BMI  -DHAVE_LZCNT -DHAVE_UINT128_EXTENSION
-DROCKSDB_SUPPORT_THREAD_LOCAL -isystem third-party/gtest-1.8.1/fused-src' >>
rocksdb.pc
install -d /home/atr/local//lib
install -d /home/atr/local//lib/pkgconfig
for header_dir in `find "include/rocksdb" -type d`; do \
       install -d //home/atr/local//$header_dir; \
done
for header in `find "include/rocksdb" -type f -name *.h`; do \
       install -C -m 644 $header //home/atr/local//$header; \
done
for header in  ; do \
       install -d //home/atr/local//include/rocksdb/`dirname $header`; \
       install -C -m 644 $header //home/atr/local//include/rocksdb/$header; \
done
install -C -m 644 rocksdb.pc /home/atr/local//lib/pkgconfig/rocksdb.pc
  GEN      util/build_version.cc
  CC       util/build_version.o
  AR       librocksdb.a
```

```
/usr/bin/ar: creating librocksdb.a
install -d /home/atr/local//lib
  CCLD     db_bench
install -C -m 755 librocksdb.a /home/atr/local//lib
[ -e librocksdb.so.6.25.0 ] && make install-shared || :
```

Important: Do not give RocksDB compilation more than 2 or 4 threads, it will freeze your VM.

What is RTTI: Run time type information. gcc flag, -fno-rtti, disables generation of information about every class with virtual functions for use by the C++ runtime type identification features (`dynamic_cast' and `typeid'). If you don't use those parts of the language, you can save some space by using this flag. Note that exception handling uses the same information, but it will generate it as needed. The `dynamic_cast' operator can still be used for casts that do not require runtime type information, i.e. casts to void * or to unambiguous base classes.

Also we are disabling the use of jemalloc, a multicore scalable malloc implementation useful in multithreaded high performance implementations. Typically it would be helpful to debug some memory leaks or segmentation faults. Disabling it is not strictly needed, but I did have to debug some hard memory errors in my code so I am recommending to disable it for now.

What is happening in the code: We integrated a skeleton code in the RocksDB compilation that registers a function with an URI API into RocksDB library. If you browse the copied (single) file `rocks_s2fs.cc`:

```
extern "C" FactoryFunc<FileSystem> stosys_s2fs_reg;
FactoryFunc<FileSystem> stosys_s2fs_reg =
       ObjectLibrary::Default()->Register<FileSystem>(
               "s2fs:.*://.*", [](const std::string &uri,
std::unique_ptr<FileSystem> *ret_fs,
                                std::string *errmsg) {
                   ret_fs->reset(FileSystem::Default().get());
                   return ret_fs->get();
               });
```

The code only does two useful things (1) it tells the RocksDB library that the stosys_s2fs_reg factory function exists (it is compiled and linked with --as-needed and with -u flags. From the gcc man page -u symbol = Pretend the symbol symbol is undefined, to force linking of library modules to define it. (2) tell RocksDB what is the URI of our new file system for which the calls will be forwarded to our code. In this case, it is `"s2fs:.*://.*"`. It follows the RegEX syntax, hence, `s2fs:.*` (one or more parameters, here you can pass the device name), then “`://`” the separator, and the rest of the path name. In the current code, it returns the FileSystem::Default() (which is your normal POSIX backed FS). In the given framework to you, there is a S2FileSystem class defined with empty function. Please implement your functions there.

Step 2: enable compiling milestone 4 in CMakeList.txt file

After you have installed the RocksDB, you should enable compiling Milestone 4 and 5 in your CMakeLists.txt file by

```
# Project configuration specific parameters
```

```
set(STOSYS_M45 OFF)
set(STOSYS_CMAKE_DEBUG OFF)
set(STOSYS_ASAN ON)
```

Set the STOSYS_M45 to ON

```
set(STOSYS_M45 ON)
```

Then make sure to

```
# in the msc-stosys-framework directory
$ make clean
$ cmake . ← redo the CMake after modifying the CMakeFile
$ make -j
```

In case making or linking problems occur, try to clean the make and cmake build files and redo the build steps.

```
# delete all build files
$ make clean
$ rm -rf cmake-build-debug/ CMakeFiles/ CMakeCache.txt cmake_install.cmake
Makefile
```

Step 3: Enable your file system implementation (S2FileSystem) and complete the file system code

You are given a code base that compiles. It compiles a testing framework (m45_main.cc) and two file system implementations (1) a DummyFSForward file system and (2) S2FileSystem.cc file system. The DummyFS file system just forwards all the calls to the default file system implementation which is your normal POSIX file system wherever you are running the RocksDB. You can use this file system to see what is being passed, what semantics are expected, and how the calls should behave, in case it is not immediately clear from the description. In case you want to browse the RocksDB POSIX file system implementation it is at: https://github.com/facebook/rocksdb/blob/main/env/fs_posix.cc (see the class PosixFileSystem : public FileSystem implementation and calls).

When you run the given code after compiling from the last step, you will see something like

```
$ sudo ./bin/m45 -p s2fs:nvme0n1:///tmp/testdb/ -e 1 -k 10 -v 10
================================================================
Welcome to milestone 4/5, which is about integration into RocksDB (also:
congratulations for bearing with us so far!)
================================================================
================================================================
test_uri s2fs:nvme0n1:///tmp/testdb/ , shadow uri posix:///tmp/shadowdb
entries 1, each entry ksize 10 bytes, vsize 10 bytes, readonly verify : 0 deleteall
: 0 single : 0
=================================================================
Opening database at posix:///tmp/shadowdb with uri posix and the db_path as
```

```
/tmp/shadowdb
Environment from URI posix:///tmp/shadowdb for FS Posix File System
## Database opened at posix:///tmp/shadowdb db name is /tmp/shadowdb , attached FS
is --> Posix File System<--
Opening database at s2fs:nvme0n1:///tmp/testdb/ with uri s2fs:nvme0n1 and the
db_path as /tmp/testdb/
Initialization uri is s2fs:nvme0n1:///tmp/testdb/ and errmsg:
DummyFSForward to Posix File System allocated
File system for s2fs:nvme0n1:///tmp/testdb/ is DummyFSForward to Posix File System
Environment from URI s2fs:nvme0n1:///tmp/testdb/ for FS DummyFSForward to Posix
File System
 call_seq: 0 tid: 8229765406951192747  func: GetAbsolutePath line: 350
 call_seq: 1 tid: 8229765406951192747  func: CreateDirIfMissing line: 236
 call_seq: 2 tid: 8229765406951192747  func: FileExists line: 173
 call_seq: 3 tid: 8229765406951192747  func: NewLogger line: 341
[...]
 call_seq: 34 tid: 8229765406951192747  func: NewWritableFile line: 99
 call_seq: 35 tid: 8229765406951192747  func: NewSequentialFile line: 69
 call_seq: 36 tid: 8229765406951192747  func: RenameFile line: 268
 call_seq: 37 tid: 8229765406951192747  func: GetChildren line: 187
 call_seq: 38 tid: 8229765406951192747  func: GetChildren line: 187
## Database opened at s2fs:nvme0n1:///tmp/testdb/ db name is /tmp/testdb/ ,
attached FS is --> DummyFSForward to Posix File System<--
Preparing the map to insert values for 1 entries, max size 20
a testmap is filled with 1 of max_size 20 each
All values inserted, number of entries 1, expected size stores would be 20 bytes
Starting data reading from the shadow db at posix:///tmp/shadowdb | single 0
*********************************************
OK: all 1 values matched successfully
 *********************************************
 call_seq: 39 tid: 8229765406951192747  func: GetChildren line: 187
 call_seq: 40 tid: 8229765406951192747  func: UnlockFile line: 321
database(s) closed, test is done OK
```

You can see that the dummy FS just displays a line with the call sequence and thread id, and just forwards the calls to the underlying POSIX file system.

Now you should enable your own S2FileSystem implementation by changing rocks_s2fs.cc:

```
if(false){
   S2FileSystem *z = new S2FileSystem(uri, true);
   ret_fs->reset(z);
} else {
   //DummyFSForward is forwarding implementation - I should be left in peace
   class DummyFSForward *m = new DummyFSForward();
   ret_fs->reset(m);
}
```

Change false to true in the if condition to get calls to your file system (sorry I did not have a much smarter way to do it for now). The constructor of the file system is filled out for you to give you an example of how parameters could be passed to it from URI. The constructor currently initializes the ZNS device using the libstosys API. The majority of the file system calls are left unimplemented. These are the ones that you have to fill out.

What choices you have to make:

1. What is the file system layout
2. How do you save the file system metadata and data
    a. Feel free to choose a log-structured file system or a simple FAT/inode style.
    b. What is the largest file size or file name you can support
    c. How do you keep track of blocks allocated
    d. How do you keep track of file structure
3. How do you plan to shutdown and resume
4. We will ask you what is the largest key-value configuration that you can run on your file system implementation and points will be awarded in a ranked manner. More details to follow later.

Hints:

1. You do not have to implement all the calls. Populate calls on demand when they are called.
2. RocksDB is multithreaded, so keep appropriate locks

Step 4: Run the test program

Here is an example of how you can run the test program:

```
$ sudo ./bin/m45 -p s2fs:nvme0n1:///tmp/atr1/
Welcome to milestone 4/5, which is integration into RocksDB
=========================================================================
test_uri s2fs:nvme0n1:///tmp/atr1/ , shadow uri posix:///tmp/shadowdb
entries 1, each entry ksize 10 bytes, vsize 90 bytes, readonly verify : 0 deleteall
: 0
=========================================================================
Opening database at posix:///tmp/shadowdb with uri posix and the db_path as
/tmp/shadowdb
Environment from URI posix:///tmp/shadowdb for FS Posix File System
## Database opened at posix:///tmp/shadowdb db name is /tmp/shadowdb , attached FS
is --> Posix File System<--
Opening database at s2fs:nvme0n1:///tmp/atr1/ with uri s2fs:nvme0n1 and the db_path
as /tmp/atr1/
[...]

Preparing the map to insert values for 1 entries, max size 100
a testmap is filled with 1 of max_size 100 each
All values inserted, number of entries 1, expected size stores would be 100 bytes
Starting data reading from the shadow db at posix:///tmp/shadowdb | single 0
**********************************************
OK: all 1 values matched successfully
 **********************************************
database(s) closed, test is done OK
```

What does the test program (m45) do and what parameters does the test program takes:

```
$ sudo ./bin/m45 -h
Welcome to milestone 4/5, which is integration into RocksDB
```

```
options: -p val -t val -r -s val -e val -d -h

 -p test_db_uri, example: -p posix:///tmp/testdb or -p s2fs:nvme0n1:///tmp/atr1/ ,
delimited is :// (3 char)

 -t is test/shadow DB which will be populated in parallel with the same keys as the
testdb and is used when using roverify in ro mode

-r says do a readonly verification from the shadowdb, it is assumed that the shadow
db is primed before by running this program once

-k integer: max size of the key

-v integer: max size of the value

-e integer: total number of entries to test for

-D delete the databases at the test and shadow locations

-S do a single DB test on the test_db with -p, no comparison

-d show debugging information

-h show this help
```

Just like in milestone 3, the test program here creates a shadowDB on the side using the default POSIX file system that you are running. Any key-value that is inserted in the KV store running on top of the S2FS file system, is also inserted in the POSIX file system. In the end they are scanned and matched. Some options that you want to know about

- -S : does not do shadow comparison, just simply inserts and reads the value back. Can be useful if you want to debug your S2FS code with a simple test program. You can pass -e 1 and just insert and test one kv tuple.
- -k, -v and -e control the amount of data that is inserted. These are in bytes, and randomly generated strings are inserted as key-value values.
- -r does a read-only verification of the test db against a shadow db. This is used for persistence tests. It assumes that there have been previous runs through which data has been inserted in the two databases.

Specific goals for M4:

1. Design and implement a file system API implementation on top of ZNS device in RocksDB KV store using your FTL-enabled device
2. Pass the test conditions. Please test what is the maximum number of keys you can test with 10+10 bytes K+V tests. Naturally the larger key or value sizes means more space on the ZNS device.
3. Upload the code on Canvas as a zip file with the format : milestone4-groupNN.zip (or tar or whatever) Check the upload deadline on canvas.

4.5 Milestone 5 (week 7) : Wake up, Neo

Type: group project

Maximum points: 10
What to do: Make everything persistent (FTL and FS), code upload and the final interview
Where: The final interview on the campus
Deadline: see the course canvas page

The last milestone is simple to explain. Make everything persistent. That means the value and data that I have stored in my previous RockDB test runs, I should be able to read and verify their content. For this milestone, you need to reserve zones to write out metadata at the FTL and file system level, and then resume next time when the system starts.

How do I test if things are working? We are going to use the same testing setup that we used before. For example, we can insert multiple 10,000 key-values in the test and shadow db by executing the following command, and then verify them as

```
$ sudo ./bin/m45 -p s2fs:nvme0n1:///tmp/atr1/ -e 10000 -k 10 -v 10
…
Starting data reading from the shadow db at posix:///tmp/shadowdb | single 0
**********************************************
OK: all 10000 values matched successfully
 **********************************************
database(s) closed, test is done OK

$ sudo ./bin/m45 -p s2fs:nvme0n1:///tmp/atr1/ -e 10000 -k 10 -v 10
…
Starting data reading from the shadow db at posix:///tmp/shadowdb | single 0
**********************************************
OK: all 20000 values matched successfully
 **********************************************
database(s) closed, test is done OK

$ sudo ./bin/m45 -p s2fs:nvme0n1:///tmp/atr1/ -e 10000 -k 10 -v 10
…
Starting data reading from the shadow db at posix:///tmp/shadowdb | single 0
**********************************************
OK: all 30000 values matched successfully
 **********************************************
database(s) closed, test is done OK


# now we run the benchmark in verify only mode as
$ sudo ./bin/m45 -p s2fs:nvme0n1:///tmp/atr1/ -r
…
Starting data reading from the shadow db at posix:///tmp/shadowdb | single 0
**********************************************
OK: all 30000 values matched successfully
 **********************************************
database(s) closed, test is done OK
```

Hint: make sure you understand what is the current state of the DB and the ZNS device between the last set of writes and the new one. Whenever unsure, delete all the old databases and reset the ZNS device to start from scratch. For example, you may want to do something like:

```
# shadow db
$ rm -rf  /tmp/shadowdb

# another db location used in the code, this is the default testdb
$ rm -rf  /tmp/testdb

# reset the ZNS device
$ sudo nvme zns reset-zone -a /dev/nvme0n1
```

Sometimes even after a reset the ZNS device can return the old content. After the rest the zones are in EMPTY state, and read on an EMPTY state is not defined. Hence, any garbage content can be returned. In this case it happened to be the last value. So when this happens, just use nvme-cli command to write out on the desired zones, or LBA to zero them out.

You may also want to browse the file content and sizes in the POSIX /tmp location to see how it should look on your file system on ZNS for comparison.

Specific goals for M5:

1. Implement persistency so that data and context can be saved across multiple runs
2. Pass multiple run setup as shown above
3. Make sure that you pass the test code with some reasonably high value of key-values
4. Upload the code on Canvas as a zip file with the format : milestone5-groupNN.zip (or tar or whatever) Check the upload deadline on canvas.
5. **Show up for the interview**, give demo, and explain you code

5. Bonus Ideas and Timeline

For the brave (or Masochists) among us we are not yet satisfied with the level of pain offered in this project, we also do offer bonuses that you can do within the project framework. Each of these bonus ideas is worth 10 points. There are no partial marks for the bonus. **The maximum points that you can achieve in bonuses is limited to 20.** You need to implement the bonus, and demonstrate that it works by implementing additional code that exercises your bonus feature implementation, and show conclusively that your implementation works. The burden of proof of your bonus code is on you, unfortunately.

Timeline and Logistics: send us (via email) the bonus plan and the number of experiments, and results that you intend to show. This should not be more than 1-2 pages. The idea of this write up is to tell us up front what you will be demonstrating and how you plan to show that your bonus works. We can either give a go ahead, or ask you to delete some or include some more stuff. If it looks sufficient, we will green light and give you a timeslot for demo and explanation. *The deadline for getting a bonus greenlight is milestone 3 interview (IMPORTANT)*. The bonus demo date will be after the milestone 5 interview.

Here are some ideas, however, we are open to ideas from your side as well (many of these ideas can be expanded into a MSc thesis work as well)

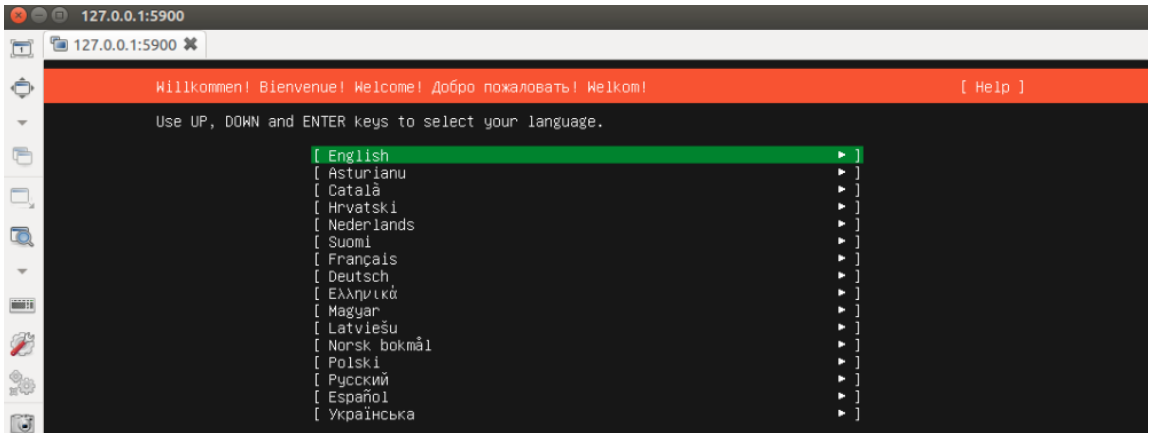
1. Deploy your code on real NVMe ZNS devices and benchmark. We have 2x NVMe ZNS devices (4TB in size). First come first serve here. We might have more but not yet confirmed.
2. Integrate another NVMe access backend (other than libnvme), for example, SPDK or xNVMe libraries. (PS~ these are asynchronous libraries, not blocking synchronous like libnvme).
3. ZNS emulation in QEMU is not persistent. If you reboot your VM then the ZNS devices come back all clean. Make it persistent.
4. Showcase multi-tenancy of your implementation by running two (or multiple) parallel RocksDB threads with the expectation that the performance should scale linearly with the number of cores.
5. Run db_bench on your implementation, and compare and explain results with a run from a file system like F2FS or ext4 and your implementation
6. Integrate and run YCSB benchmark (https://github.com/brianfrankcooper/YCSB) on RocksDB on top of your Milestone 5. YCSB already has a RocksDB plugin, you need to integrate the whole thing and show that it works, and produce experimental results.
7. Integrate your FTL-device into another KV or SQL engine (MySQL, FoundationDB, ScyllaDB).
8. Your idea? I am curious to hear it.

For more information please ask us in the lab hours.

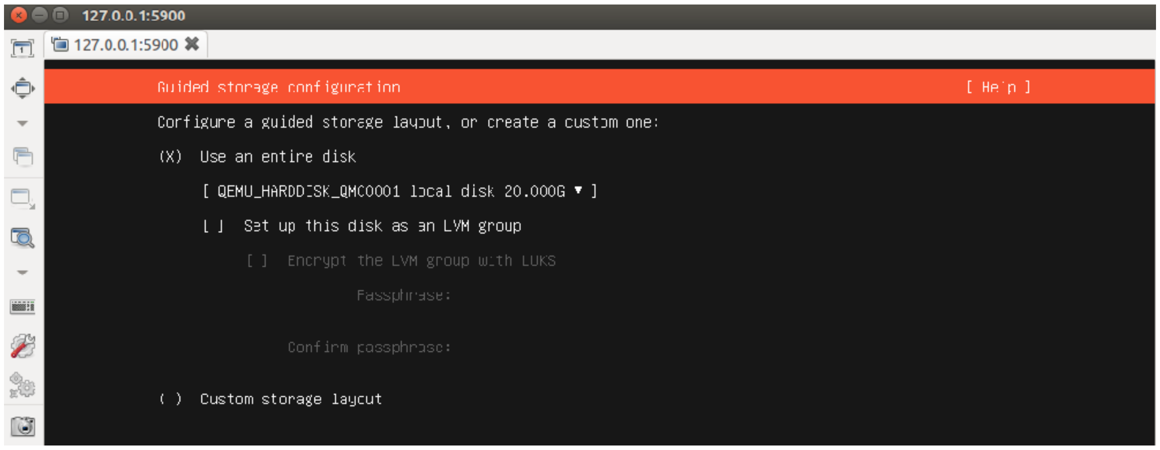
Appendix

A. Step by step guide for setting up Ubuntu installation

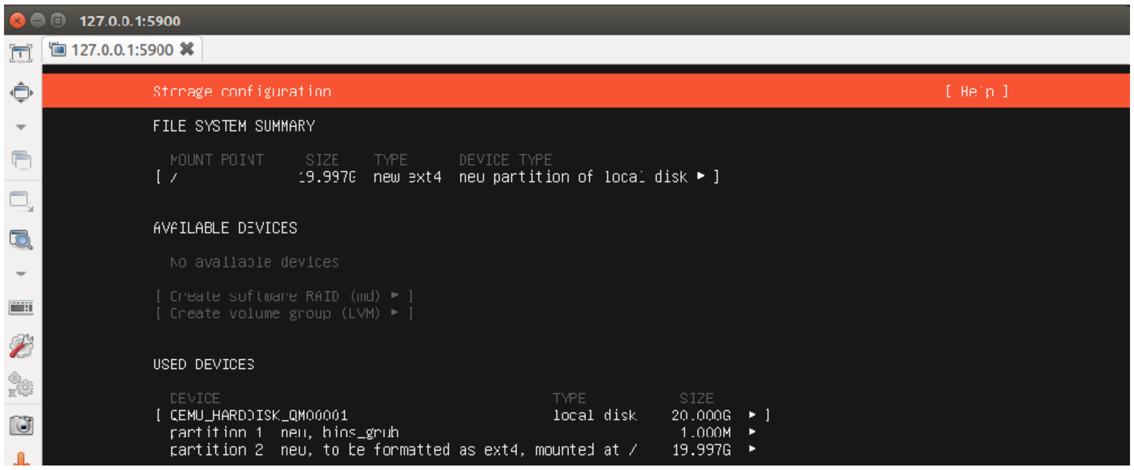
Here are “some” sequences of print screens that you should see when installing the Ubuntu OS on your image.



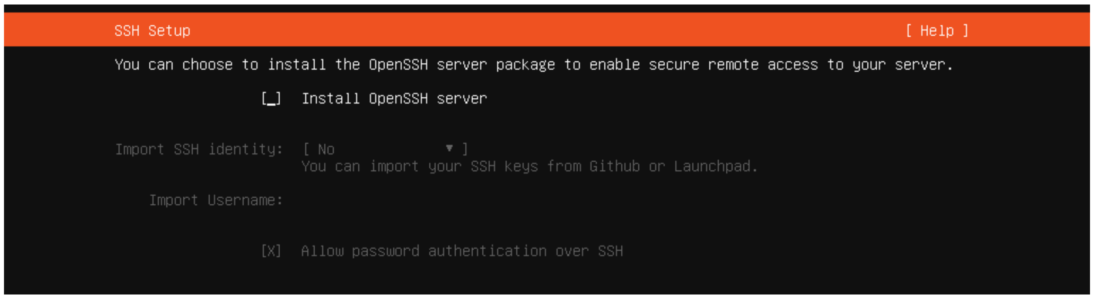
```
Willkommen! Bienvenue! Welcome! Добро пожаловать! Welkom!                    [ Help ]

Use UP, DOWN and ENTER keys to select your language.

                  [ English                                      ▸ ]
                  [ Asturianu                                    ▸ ]
                  [ Català                                       ▸ ]
                  [ Hrvatski                                     ▸ ]
                  [ Nederlands                                   ▸ ]
                  [ Suomi                                        ▸ ]
                  [ Français                                     ▸ ]
                  [ Deutsch                                      ▸ ]
                  [ Ελληνικά                                     ▸ ]
                  [ Magyar                                       ▸ ]
                  [ Latviešu                                     ▸ ]
                  [ Norsk bokmål                                 ▸ ]
                  [ Polski                                       ▸ ]
                  [ Русский                                      ▸ ]
                  [ Español                                      ▸ ]
                  [ Українська                                   ▸ ]
```



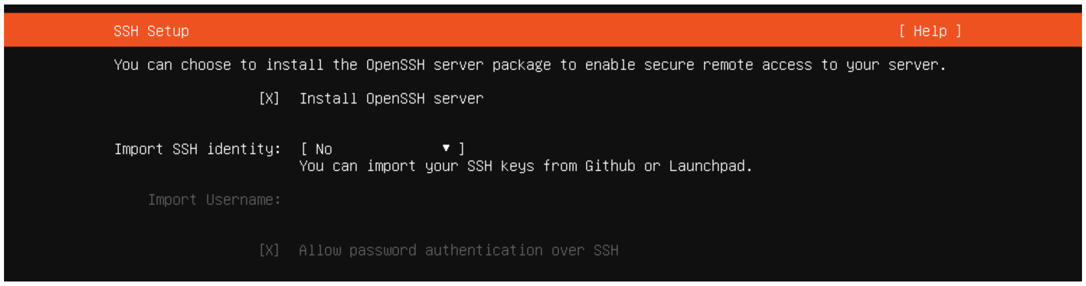
```
Guided storage configuration                                                 [ Help ]

Configure a guided storage layout, or create a custom one:

(X)  Use an entire disk

        [ QEMU_HARDDISK_QM00001 local disk 20.000G ▾ ]

        [ ]  Set up this disk as an LVM group

             [ ]  Encrypt the LVM group with LUKS

                          Passphrase:


                  Confirm passphrase:


( )  Custom storage layout
```



```
Storage configuration                                                        [ Help ]

FILE SYSTEM SUMMARY

  MOUNT POINT     SIZE    TYPE      DEVICE TYPE
[ /              19.997G  new ext4  new partition of local disk ▸ ]


AVAILABLE DEVICES

  No available devices

[ Create software RAID (md) ▸ ]
[ Create volume group (LVM) ▸ ]


USED DEVICES

  DEVICE                                          TYPE          SIZE
[ QEMU_HARDDISK_QM00001                           local disk    20.000G  ▸ ]
  partition 1  new, bios_grub                                    1.000M  ▸
  partition 2  new, to be formatted as ext4, mounted at /       19.997G  ▸
```

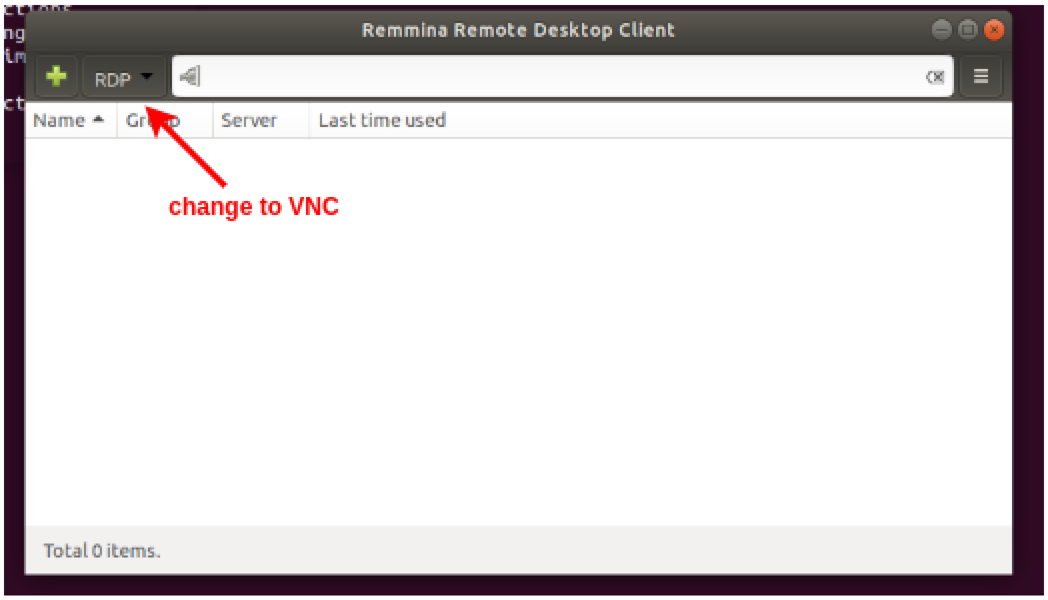


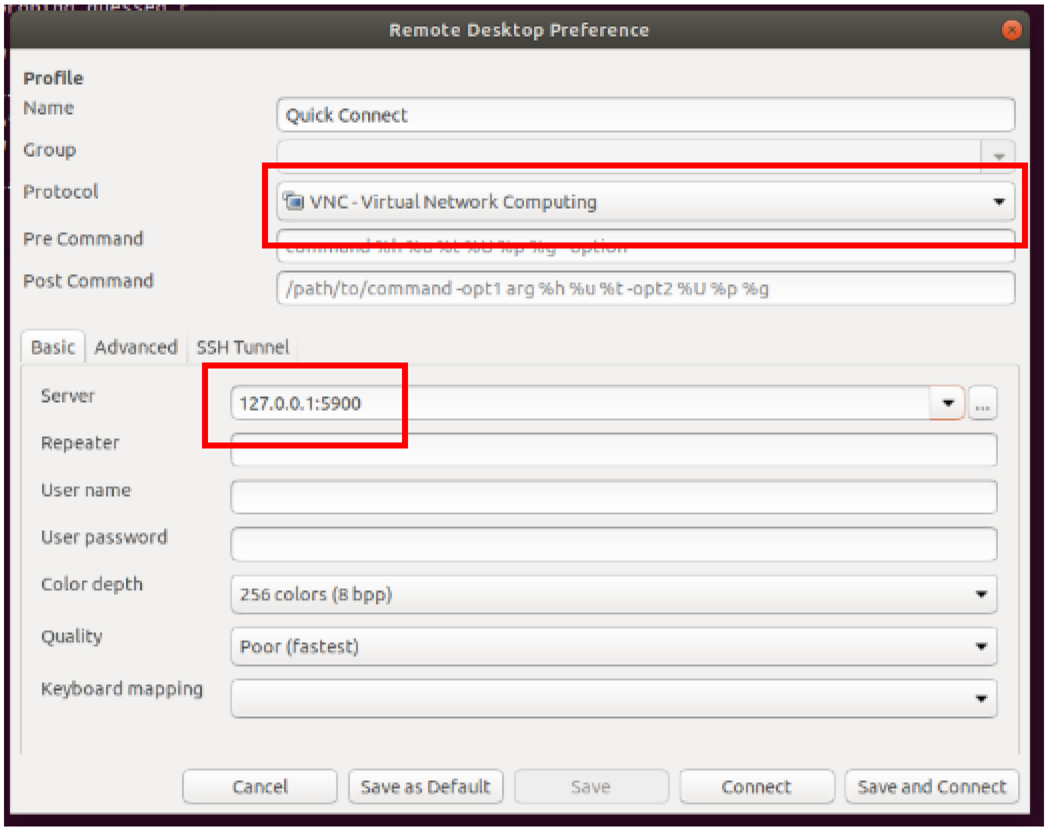
```
SSH Setup                                                                    [ Help ]
You can choose to install the OpenSSH server package to enable secure remote access to your server.
                    [ ]  Install OpenSSH server

Import SSH identity:  [ No             ▾ ]
                      You can import your SSH keys from Github or Launchpad.

     Import Username:

                    [X]  Allow password authentication over SSH
```



```
SSH Setup                                                                    [ Help ]
You can choose to install the OpenSSH server package to enable secure remote access to your server.
                    [X]  Install OpenSSH server

Import SSH identity:  [ No             ▾ ]
                      You can import your SSH keys from Github or Launchpad.

     Import Username:

                    [X]  Allow password authentication over SSH
```

B. Setup of VNC through Remmina





Remote Desktop Preference

Profile

Name	Quick Connect
Group	
Protocol	VNC - Virtual Network Computing
Pre Command	command %h %u %t %U %p %g --option
Post Command	/path/to/command -opt1 arg %h %u %t -opt2 %U %p %g

Server	127.0.0.1:5900
Repeater	
User name	
User password	
Color depth	256 colors (8 bpp)
Quality	Poor (fastest)
Keyboard mapping	

C. Running inside Virtualbox, VirtualMachineManager or VMWare

If you run Virtualbox, VirtualMachineManager or VMWare the command to launch qemu-system will fail with

```
ioctl(KVM_CREATE_VM) failed: 16 Device or resource busy
qemu-system-x86_64: failed to initialize KVM: Device or resource busy.
```

This is due to a limitation in CPU virtualization extensions Intel VT-x and AMD-V. You can either disable all other virtualization software or remove -enable-kvm However, this will be significantly slower and you will need to specify a cpu model.

Also see: https://ostechnix.com/how-to-enable-nested-virtualization-in-virtualbox/