COMP3301 2023 Assignment 3 - OpenBSD QCOW Kernel Driver

- Due: Week 13
- $Revision: 501 $

1 OpenBSD QCOW Kernel Driver

This assignment extends the OpenBSD kernel to add support for using QCOW2 disk images as block devices.

This is similar to the existing functionality provided by the vnd(4) driver, which supports using files containing a disk image as a block device.

The purpose of this assignment is to excercise concepts of low level disk operations and page tables in an operating system kernel environment.

This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students, and discuss OpenBSD and its APIs in general terms. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code will be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion (outside of the base code given to everyone), formal misconduct proceedings will be initiated against you. If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code. You should read and understand the statements on student misconduct in the course profile and on the school web-site: https://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism

2 Background

From a high level point of view, a physical disk device presents a sequence of bytes that can be written to or read from, with the ability to quickly seek to an arbitrary position and read and write at that point. Note that this is a simplification that ignores that disks address and provide access to blocks of bytes, not individual bytes.

A file on most operating systems offers similar features, ie, a sequence of bytes that can be accessed by address. Because of these similarities it is possible for an operating system to provide a common set of operations on both files and disks (eg, open, close, read, write, seek, etc) and allow them to be used interchangably. For example, you could use `tar` to write an archive to a file in a filesystem, or directly to a disk device. `dd`, `cp`, `cat`, etc can read the bytes from a raw disk into a file or visa versa.

However, operating systems generally provide extra functionality on top of disk devices such as the ability to partition disks and mount filesystems from them.

2.1 vnd(4)

The vnd(4) driver in OpenBSD provies a "disk-like interface to a file". This means the OpenBSD kernel can open a file and present it as a block device to the rest of the system, which in turn allows for the creation and use of

filesystems on these disk images.

However, the vnd(4) driver only supports using raw disk images as backing files.

This means that there's a one to one mapping of data offsets for data in the vnd disk device and the byte offset of that data in the underlying file. This makes the implementation very simple, with the downside that the backing file has to be the same size as the disk vnd is presenting. If you have a 32G disk image, the file will be 32G regardless of how much data is actually stored inside a filesystem mounted on top of it.

Similar functionality exists in the loop driver in Linux, and the lofi driver in Solaris and Illumos.

2.2 QCOW

QCOW is short for "QEMU Copy On Write", and it is a file format for disk images used by QEMU. A defining feature of QCOW files is that they allocate space and extend the size of the file backing a disk image on demand. This is different to raw disk images where the backing file has to pre-allocate space for the entire disk image up front.

QCOW also supports using a read only file as a base image and writing changes to a separate file. When used by a hypervisor, this allows for thin provisioning of virtual machines where only changes made by each virtual machine are stored rather than repeated copies of disks.

QEMU has developed several versions of the QCOW file format. The current version, version 3, is documented in qcow2.txt in their source repository. QCOW 3 is largely an extension to QCOW 2.

Due to the popularity of QEMU, it is common for disk images to be distributed as QCOW files.

3 Specifications

You will be extending the OpenBSD kernel to add support for using QCOW files as a backend for a qcow(4) virtual block device. qcow(4) is roughly based on vnd(4).

Boilerplate code for the device entry points and command line utility will be provided, but you will be implementing the handling of the file and the QCOW file format within the provided kernel driver.

Only a subset of the QCOW functionality listed in the QEMU specification of the file format is required.

The following functionality is required:

- read support
- write support

The following is not required:

- separate backing files
- copy-on-write
- snapshots
- compression
- encryption
- any incompatible, compatible, or autoclear features

In addition to supporting the QCOW file format, the kernel should implement the following:

- deny detaching qcow files when the disk is open unless the force flag is passed to the `QCOWIOCDETACH` ioctl
- return the name of the qcow file the device was attached to for the `QCOWIOCFNAME` ioctl.
- populate a `struct stat` for the currently attached qcow file for the `QCOWIOCSTAT` ioctl.

3.1 Apply the diff

- Fetch https://stluc.manta.uqcloud.net/comp3301/public/2023/a3-base.patch
- Create an a3 branch
  - 'git checkout -b a3 openbsd-7.3
- Apply the base patch to the a3 branch
  - `git am /path/to/a3-base.patch` in /usr/src
- Build the kernel
  - `cd /usr/src/sys/arch/amd64/compile/GENERIC.MP`
  - `make obj`
  - `make config`
  - `make -j 5`
  - `doas make install`
- Reboot into the kernel
  - `reboot`
- `make obj` in /usr/src
- `doas make includes` in /usr/src/include
- make qcowctl(8)
  - `cd /usr/src/usr.sbin/qcowctl`
  - `make`
  - `doas make install`

3.2 QCOW disk I/O

Reads and writes against the qcow block device should be mapped to vnd_rdwr() against the QCOW backing file, as per the qcow2.txt specification.

3.3 `ioctl` interface

The following ioctls should only work on the raw partition of the character device associated with each qcow disk. Except for `QCOWIOCATTACH`, they should only work when a qcow disk is attached to a backing file.

3.3.1 `QCOWIOCATTACH`

Specify the QCOW file to attach as a block device, and parameters for using the disk in the kernel. The `qcow_attach` struct contains the following fields:

- `qc_file`

The name of the qcow file to attach a qcow disk to.

- `qc_readonly`

When set to zero the qcow disk can be written to. If the qcow file is read-only, the qcow disk should fail to attach.

If `qc_readonly` is non-zero then the qcow disk should be read-only. If qcow file should be opened read-only with this option set.

- `qc_secbits`

The size of the qcow disk sectors the driver should present to the kernel, expressed as an exponent. The size of the disk sectors in bytes can be calculated as `1 << qc_sebits`, eg, if the value is 9, then `1 << 9` is 512, meaning the qcow disk sector size is 512 bytes.

If this value is 0 then use a default secbits value of 9, or 512 byte sectors.

This is independent of the size of the QCOW file cluster size.

3.3.2 QCOWIOCDETACH

This ioctl requests the block device be detached, and the backing file closed.

If the disk is in use, the request should fail with `EBUSY` unless the unsigned int ioctl argument is set to a non-zero value. A non zero value requests the forced removal of the block device and close of the backing QCOW file.

3.3.3 QCOWIOCFNAME

This ioctl requests the name of the QCOW file used for the currently attached block device. The name should be the same as what was passed as the filename in the `QCOWIOCATTACH` ioctl.

3.3.4 QCOWIOCSTAT

This ioctl is equivalent to an `fstat(2)` system call against the backing file.

3.4 Code Style

Your code is to be written according to OpenBSD's style guide, as per the `style(9)` man page.

An automatic tool for checking for style violations is available at https://cstuc.manta.uqcloud.net/comp3301/public/2022/cstyle.pl. This tool will be used for calculating your style marks for this assignment.

3.5 Compilation

Your code for this assignment is to be built on an `amd64` OpenBSD 7.3 system identical to your course-provided VM.

The following steps must be succeed:

- `make obj; make config; make` in src/sys/arch/amd64/compile/GENERIC.MP
- `make obj; make includes` in src
- `make obj; make; make install` in src/usr.sbin/qcowctl

The existing `Makefiles` in the provided code are functional as-is, but may need modification as part of your work for this assignment.

Note that the existing Makefile ensures the `-Wall` flag is passed to the compiler, as well as a few other warning and error-related flags.

3.6 Provided code

The provided code which forms the basis for this assignment can be downloaded as a single patch file at:

https://stluc.manta.uqcloud.net/comp3301/public/2023/a3-base.patch

You should create a new a3 branch in your repository based on the `openbsd-7.3` tag using `git checkout`, and then apply this base patch using the `git am` command:

```
$ git checkout -b a3 openbsd-7.3
$ ftp -o /tmp/a3-base.patch https://stluc.manta.uqcloud.net/comp3301/public/2023/a3-base.patch
$ git am < /tmp/a3-base.patch
$ git push origin a3
```

3.6.1 `qcowctl`

The patch includes source for a `qcowctl` utility that uses the ioctls defined above to control qcow devices. It is similar to `vnconfig`.

The source can be found after the patch is applied in `src/usr.sbin/qcowctl`.

3.6.2 `MAKEDEV`

The `MAKEDEV` script that can be installed in `/dev/MAKEDEV` has been updated to support creation of qcow disk devices nodes. Once installed, you can run `MAKEDEV qcow0` in `/dev` to create all the devices nodes for qcow0 for example.

if you encounter an error "no such device called qcow", copy the MAKEDEV file from `/usr/src/etc/etc.amd64/` to the `/dev` folder and then run it.

3.7 Reflection

Provide a reflection on your implementation, by briefly answering the following questions:

1. Describe the steps you took or draw a flowchart.
2. Describe an error that you encountered.
3. Describee how the error was debugged.
4. Describe how the bug was solved.

Upload both pdf and your answers it as a pdf to the Blackboard a3 reflection submission. Page length is a maximum 2 pages or less.

Pdf name must be your STUDENT_NUMBER_a3.pdf

Note this is your XXXXXXXX ID number and not sXXXXXXX login.

4 Submission

Submission must be made electronically by committing to your Git repository on `source.eait.uq.edu.au`. In order to mark your assignment the markers will check out the a3 branch from your repository. Code checked in to any other branch in your repository will not be marked.

As per the `source.eait.uq.edu.au` usage guidelines, you should only commit source code and Makefiles.

Your a3 branch should consist of:

- The openbsd-7.3 base commit
- The a3-base.patch commit
- Commit(s) for adding the required functionality

4.1 Marking

Your submission will be marked by course tutors and staff, during an in-person demo with you, at your lab session during the due week. You must attend your session, in-person, otherwise your submission will not be marked. Online attendence, e.g. zoom, is not permitted.

5 Testing

The dd(1) command can be used to transfer bytes from/to block driver devices. It is recommended to attempt a read operation before a write operation.