SPDK fsdev Introduction

Anton Nayshtut, SW Engineer | 2024-02-01

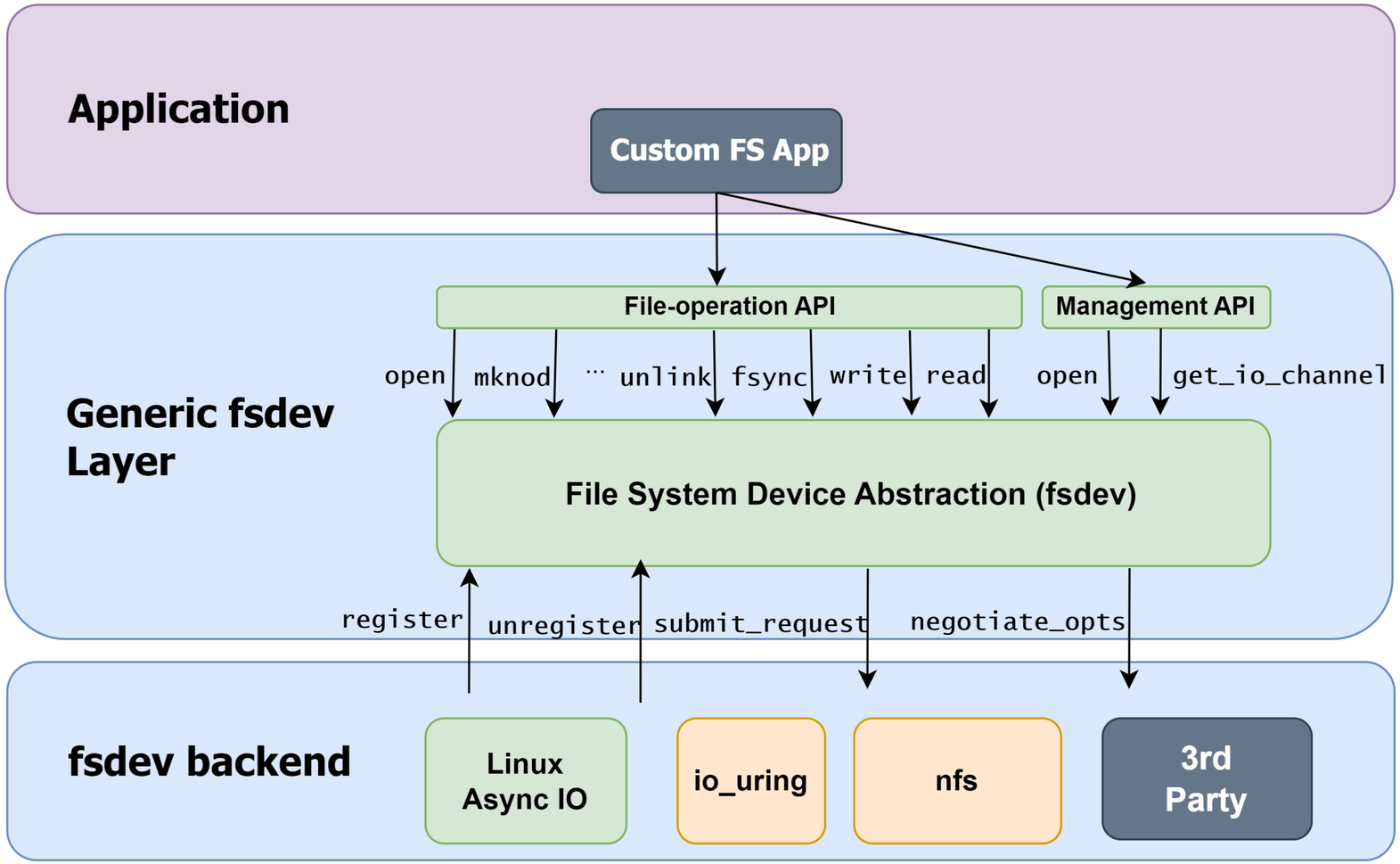
SPDK File System Device

bdev for the File Systems

- The “S” in SPDK means **Storage**
  - *bdev* represents the block storage layer
  - *fsdev* represents the file system layer
- *fsdev* is inspired by *bdev* and shares the same ideology (concepts, partitioning, modules etc.)
- The SPDK file system device layer (hereafter – *fsdev)*, is a C library intended to be equivalent to the operating system file system layer.
- Provides the following functionality:
  - A pluggable module API for implementing file system devices that interface with different types of file systems.
  - Driver modules
    - The 1st one - Linux/Posix AIO implementing a bridge to a local folder
  - An application API for enumerating and claiming SPDK file system devices and then performing file operations (lookup, forget, getattr, setattr, mknod, read, write, unink, etc.) on those devices
  - Facilities to stack file system devices to create complex I/O pipelines
  - Configuration of file system devices via JSON-RPC
  - Multiple, lockless queues for asynchronous handling of file operations
- Like *bdev*, the *fsdev* module creates abstraction layer that provides common API for all devices. User can use available fsdev modules or create own module with any type of device underneath

SPDK fsdev Architecture

High Level



SPDK fsdev API

bdev-like but for the file ops

- Management SPDK `fsdev` subsystem API – `bdev`-like, lot of stuff copied from there
  - `spdk_fsdev_initialize/spdk_fsdev_finish`
  - `spdk_fsdev_get_by_name`
  - `spdk_fsdev_open/spdk_fsdev_close`
  - `spdk_fsdev_get_io_channel`
  - etc.
- File System specific file-operation level `fsdev` API inspired by the libfuse's low-level API
  - `spdk_fsdev_op_lookup`
  - `spdk_fsdev_op_forget`
  - `spdk_fsdev_op_getattr/spdk_fsdev_op_setattr`
  - `spdk_fsdev_op_readlink`
  - `spdk_fsdev_op_symlink`
  - `spdk_fsdev_op_read`
  - `spdk_fsdev_op_write`
  - `spdk_fsdev_op_release`
  - `spdk_fsdev_op_fsync`
  - `spdk_fsdev_op_mkdir`
  - `spdk_fsdev_op_opendir`
  - etc.

SPDK fsdev Module

How to implement a new file system driver or a virtual fsdev

- An `fsdev` module can implement
  - a new user space file system driver – either generic (NFS, SMB etc.) or custom
  - a virtual module – encryption, QoS etc.
- Currently we only have one module implemented – AIO, implementing a Linux AIO-based bridge to a local folder
  - based on the libfuse's `passthrough_ll.c`
- The next step could be an `io_uring`-based `fsdev` module (a performance improvement)
- Some generic File Systems may come later:
  - `NFS`
  - `SMB`
  - `?`
- Customers can extend the SPDK `fsdev` functionality by implementing a custom `fsdev` module

Use case - QEMU

fsdev as a virtio-fs backend

- QEMU supports user-land virtio-pci devices
  - via the virtio-specific vhost-user API (old way)
  - via the generic vfio-user API (new way)
- virtiofsd-rs Rust daemon is currently responsible for emulating virtio-fs devices
- *spdk-tgt* with *fsdev* can do the same!
  - Problem: virtiofs speaks FUSE (structs, requests, responses), not aware of the file ops ☹
  - Solution: **FUSE dispatcher**! ☺
- `fuse_dispatcher` is an auxiliary library that implements the FUSE ⇔ `fsdev` API translation
  - iovec-based API according to virtio-fs spec
  - the iovecs contain the FUSE-specific structures (IN and OUT)
  - the `fuse_dispatcher`
    - parses the FUSE requests
    - calls the corresponding file-operation level `fsdev` APIs
    - format the FUSE responses
    - handles the completion
  - A minimalistic, pure functional API
    - `spdk_fuse_dispatcher_create`/`spdk_fuse_dispatcher_delete`
    - `spdk_fuse_dispatcher_sumbit_request`

QEMU with SPDK fsdev

QEMU + spdk_tgt + Virtio FS Target + FUSE dispatcher + SPDK fsdev