RocksDB with Uring(for FDP)

- Goal : RocksDB에서 raw device NVMe(/w FDP) 사용
- How To
  - cachelib에서 FDP 사용하는 방법 파악 (아마 liburing.h 기반)
  - RocksDB에서 ZNS를 사용하기 위한 파일시스템(ZenFS)에서 raw device 접근/제어 방법 파악
  - RocksDB에서 FDP를 사용하고, 이를 관리하는 파일시스템 구현
  - RocksDB Data 속성별로 FDP RU Handle 분리를 위해 Data별 I/O Access point 파악
- Others
  - SPDK 활용을 조사해보았으나 몇가지 제한사항이 있음
    - RocksDB에서 SPDK 활용을 위해 별도의 수정 필요하고, 이는 SPDK/RocksDB git에서 별도로 관리 다만, SPDK/RocksDB git에서 지속 관리되지 않고있음. 4-5년전인 RocksDB v6.15이 마지막 버전 (latest v9.2.0)
    - SPDK/nvme bdev를 통해 FDP 사용이 가능할 것으로 보이나, 정보가 많이 없음
    - 향후 MyRocks까지 고려하면 MyRocks에서 fork하는 RocksDB버전과 예상하지 못한 충돌이 발생할 가능성 있음.

Cachelib에서 FDP 적용한 과정

Study 예정

RocksDB source code memo

- gdb 사용을 위해 `make dbg` 로 빌드
  - https://github.com/facebook/rocksdb/wiki/RocksDB-Troubleshooting-Guide

RocksDB에서 File system mount 과정 (for ZNS)

Background

```
1. RocksDB는 데이터를 파일로 관리하며, 이를 위해 파일시스템이 존재함
2. raw device는 mount되어 mount point(directory)로 접근됨
3. 기본값으로는 mount된 장치의 file system을 사용하도록 io_posix(fs_posix)로 i/o 요청함
4. file_system.h에 filesystem wrapper class가 있고, 이를 상속받아 별도의 file system을 구현할 수 있음 (ZenFS 참고)
```

1. ZenFS와 같이 Filesystem을 별도로 설정하는 경우
    1. FactoryFunc AddFactory(name, func) 함수를 통해 "name"을 갖는 library를 factory에 등록
        1. <include/rocksdb/utilities/object_registery.h>
    2. zenfs_filesystem_reg과 같이 Filesystem을 Factory에 등록하기 위해 AddFactory 함수 구현
        1. <plugins/zenfs/fs/fs_zenfs.cc>
    3. env.h의 CreateFromUri를 통해 문자열로 전달받은 fs_uri를 사용하여 RocksDB에서 사용할 Filesystem 연결
        1. Env::CreateFromUri() -> FileSystem::CreateFromString(config_options, fs_uri, &fs)

4. fs_uri는 사전에 파티셔닝(mkfs) 되어있어야함 (zbdlib or zonefs)
    1. (zbdlib 예시) ./plugin/zenfs/util/zenfs mkfs --zbd= --aux_path=
5. zenfs AddFactory source code

```
  FactoryFunc<FileSystem> zenfs_filesystem_reg =
  // For old version
  #if (ROCKSDB_MAJOR < 6) || (ROCKSDB_MAJOR <= 6 && ROCKSDB_MINOR < 28)
     ObjectLibrary::Default()->Register<FileSystem>(
        "zenfs://.*", [](const std::string& uri,
std::unique_ptr<FileSystem>* f,
                         std::string* errmsg) {
  // For current version
  #else
     ObjectLibrary::Default()->AddFactory<FileSystem>(
        ObjectLibrary::PatternEntry("zenfs", false)
           .AddSeparator("://"), /* "zenfs://.+" */
        [](const std::string& uri, std::unique_ptr<FileSystem>* f,
          std::string* errmsg) {
  #endif
         std::string devID = uri;
         FileSystem* fs = nullptr;
         Status s;
```

RocksDB의 ZenFS에서 raw device 접근/제어 과정

1. RocksDB Filesystem Wrapper
    1. /home/cm/repo/rocksdb/include/rocksdb/file_system.h
2. ZenFS
    1. (raw) zbdlib_zenfs/zonefs_zenfs -> zbd_zenfs -> fs_zenfs (filesystem) 순으로 추상화됨
    2. zbd_zenfs에서 ZonedBlockDevice() 초기화 시, ZBD_Backend를 선택함 (zbdlib or zonefs)
    3. zbdlib에서 filename으로 Open()하고, Read(), Write()로 data access함. Read/Write 내부에서 zbdlib의 pread, pwrite를 호출함

```
pread(direct ? read_direct_f_ : read_f_, buf, size, pos)
pwrite(write_f_, data, size, pos)
```

참고 데이터

- zbd_zenfs.cc : ZonedBlockDevice::ZonedBlockDevice() 과정에서 backend type에 따라 zbd_be_ 결정. ZbdBackendType::kBlockDev or ZbdBackendType::kZoneFS
- zonefs_zenfs.h|59| <> class ZoneFsBackend : public ZonedBlockDeviceBackend
- zbdlib_zenfs.h|20| <> class ZbdlibBackend : public ZonedBlockDeviceBackend

```
// zbd_zenfs.h
class ZonedBlockDeviceBackend {
 public:
```

```
  uint32_t block_sz_ = 0;
  uint64_t zone_sz_ = 0;
  uint32_t nr_zones_ = 0;

 public:
  virtual IOStatus Open(bool readonly, bool exclusive,
                        unsigned int *max_active_zones,
                        unsigned int *max_open_zones) = 0;

  virtual std::unique_ptr<ZoneList> ListZones() = 0;
  virtual IOStatus Reset(uint64_t start, bool *offline,
                         uint64_t *max_capacity) = 0;
  virtual IOStatus Finish(uint64_t start) = 0;
  virtual IOStatus Close(uint64_t start) = 0;
  virtual int Read(char *buf, int size, uint64_t pos, bool direct) = 0;
  virtual int Write(char *data, uint32_t size, uint64_t pos) = 0;
  virtual int InvalidateCache(uint64_t pos, uint64_t size) = 0;
  virtual bool ZoneIsSwr(std::unique_ptr<ZoneList> &zones,
                         unsigned int idx) = 0;
  virtual bool ZoneIsOffline(std::unique_ptr<ZoneList> &zones,
                             unsigned int idx) = 0;
  virtual bool ZoneIsWritable(std::unique_ptr<ZoneList> &zones,
                              unsigned int idx) = 0;
  virtual bool ZoneIsActive(std::unique_ptr<ZoneList> &zones,
                            unsigned int idx) = 0;
  virtual bool ZoneIsOpen(std::unique_ptr<ZoneList> &zones,
                          unsigned int idx) = 0;
  virtual uint64_t ZoneStart(std::unique_ptr<ZoneList> &zones,
                             unsigned int idx) = 0;
  virtual uint64_t ZoneMaxCapacity(std::unique_ptr<ZoneList> &zones,
                                   unsigned int idx) = 0;
  virtual uint64_t ZoneWp(std::unique_ptr<ZoneList> &zones,
                          unsigned int idx) = 0;
  virtual std::string GetFilename() = 0;
  uint32_t GetBlockSize() { return block_sz_; };
  uint64_t GetZoneSize() { return zone_sz_; };
  uint32_t GetNrZones() { return nr_zones_; };
  virtual ~ZonedBlockDeviceBackend(){};
};
```

ZenFS Open command (zbdlib 사용, 향후 liburing 변경)

```
// zbdlib_zenfs.cc
IOStatus ZbdlibBackend::Open(bool readonly, bool exclusive,
                             unsigned int *max_active_zones,
                             unsigned int *max_open_zones) {
  zbd_info info;

  /* The non-direct file descriptor acts as an exclusive-use semaphore */
  if (exclusive) {
    read_f_ = zbd_open(filename_.c_str(), O_RDONLY | O_EXCL, &info);
```

```
  } else {
    read_f_ = zbd_open(filename_.c_str(), O_RDONLY, &info);
  }

  if (read_f_ < 0) {
    return IOStatus::InvalidArgument(
        "Failed to open zoned block device for read: " + ErrorToString(errno));
  }

  read_direct_f_ = zbd_open(filename_.c_str(), O_RDONLY | O_DIRECT, &info);
  if (read_direct_f_ < 0) {
    return IOStatus::InvalidArgument(
        "Failed to open zoned block device for direct read: " +
        ErrorToString(errno));
  }
```

ZenFS Read/Write command (zbdlib 사용, 향후 liburing 변경)

```
// zbdlib_zenfs.cc

int ZbdlibBackend::Read(char *buf, int size, uint64_t pos, bool direct) {
  return pread(direct ? read_direct_f_ : read_f_, buf, size, pos);
}

int ZbdlibBackend::Write(char *data, uint32_t size, uint64_t pos) {
  return pwrite(write_f_, data, size, pos);
}
```

liburing 사용방법

1. Read : 아래 링크 참고하여 sq 얻고, read command 전송, read buffer 취득하는 과정 확인 및 cachelib에서 read하는 것과 비교
    1. https://unixism.net/loti/tutorial/cat_liburing.html
2. Write : 아래 링크 참고하여 read -> write 하는 과정 이해하고, cachelib에서 write하는 것과 비교
    1. https://unixism.net/loti/tutorial/cp_liburing.html