Device mapper

http://techgmm.blogspot.kr/2012/07/writing-your-own-device-mapper-target.html
Introduction

- Generic framework to map block devices onto another
  - e.g. **LVM2**, software RAIDs, dm-crypt, multipath, ...

  ![Diagram showing File system, dm-crypt, and HDD]

- **/dev/mapper/**
  - List of virtual block devices managed by device mapper

- **dmsetup**
  - Tool to manipulate device mapper
  - Guide: [http://linuxgazette.net/114/kapil.html](http://linuxgazette.net/114/kapil.html)
Using device mapper

• **Installation**
  – `apt-get install dmsetup`

• **Example**: multipath device
  – `dmsetup create mp_test`
    
    
    ```
    0 1024 linear /dev/sda2 1024
    1024 2048 linear /dev/sdb2 0
    ```
  – **Result**: `/dev/mapper/mp_test`

• Mapped device can be nested
  – **LVM over RAID**
Writing new DM target

• DM target is a **kernel module**
  – Describes mapping behavior

• **Include files**
  – **Module**: `<linux/module.h>`, `<linux/kernel.h>`, `<linux/init.h>`
  – **Block I/O**: `<linux/bio.h>`
  – **Device mapper**: `<linux/device-mapper.h>`

• **Method functions** to implement
  – `map()`: remapping **bio** to *real* target device
  – `ctr()`: create a new virtual block device
  – `dtr()`: stop using virtual block device
### Device mapper target

#### Virtual device (/dev/mapper/mp_test)

```
struct block_device
    ->bd_disk->private_data
```

#### User defined data structure

```
struct dm_table
    <drivers/md/dm-table.c>

    int num_targets
    struct dm_target * targets
```

```
struct my_cache
    struct dm_dev * cache
    struct dm_dev * backing
    struct my_btree * metadata
```

#### Struct mapped_device

```
struct mapped_device
    <drivers/md/dm.c>

    struct dm_table * map
    struct gendisk * disk
```

```
list_head_targets
```

#### Struct dm_target

```
struct dm_target
    <linux/device_mapper.h>

    struct dm_table * table
    struct target_type * type
    void * private
    sector_t begin
    sector_t len
```

#### Struct dm_dev

```
struct dm_dev
    <linux/device_mapper.h>

    struct block_device * bdev
    fmode_t mode
    char [] name
```

#### Map device (/dev/sdb2)

```
map()
ctr()
dtr()
```

#### Underlying device (/dev/sdb2)

```
```
Map method

- static int map(struct dm_target *ti, struct bio *bio, union map_info *map_context)

- Return value
  - **DM_MAPIO_SUBMITTED**
    - submitted bio to underlying device: used submit_bio()
  - **DM_MAPIO_REMAPPED**
    - bio request is remapped: changed bi_dev, bi_sector
    - Device mapper should submit bio
  - **DM_MAPIO_REQUEUE**
    - Problem with the bio, but can finish the request with deferred retry
    - Map method will be called again
Map() method example

- For write request (bio->bi_rw & 1)
  - Write through caching
    * Allocate and initiate `bio` for *cache* device → `submit_bio()`
    * Allocate and initiate `bio` for *backing* device → `submit_bio()`
    * Return `DM_MAPIO_SUBMITTED`

- For read request
  - Cache hit
    * Update `bi_bdev = bdev_cache`, `bi_sector = cache_offset`
  - Cache miss
    * Update `bi_bdev = bdev_backing`
  - Return `DM_MAPIO_REMAPPED`
**Ctr / Dtr method**

- **static int ctr(struct dm_target *ti, unsigned int argc, char **argv)**
  - Call `dm_get_device()` to lock backing devices
  - Initialize device data structures and worker threads
  - **Return value**
    - 0 on success
    - Negative on error

- **static void dtr(struct dm_target *ti)**
  - Call `dm_put_device()` function
  - Free device data structures
Registering target

- Enables the mapping method available to the kernel

```c
int dm_register_target(struct target_type *)
```
  - struct `target_type`:
    - name = “target name”
    - version = { version }
    - module = THIS_MODULE
    - ctr(), dtr(), map(), ...
  - Returns 0 on success

```c
void dm_unregister_target(struct target_type *)
```