Inside the ext2 Filesystem & Thoughts on Writing a Filesystem Driver

Ziqin Li

System Research Association@Sichuan University

October 24, 2024

1. ext2 filesystem

2. Thoughts on Writing a Filesystem Driver

Basic Filesystem Structure

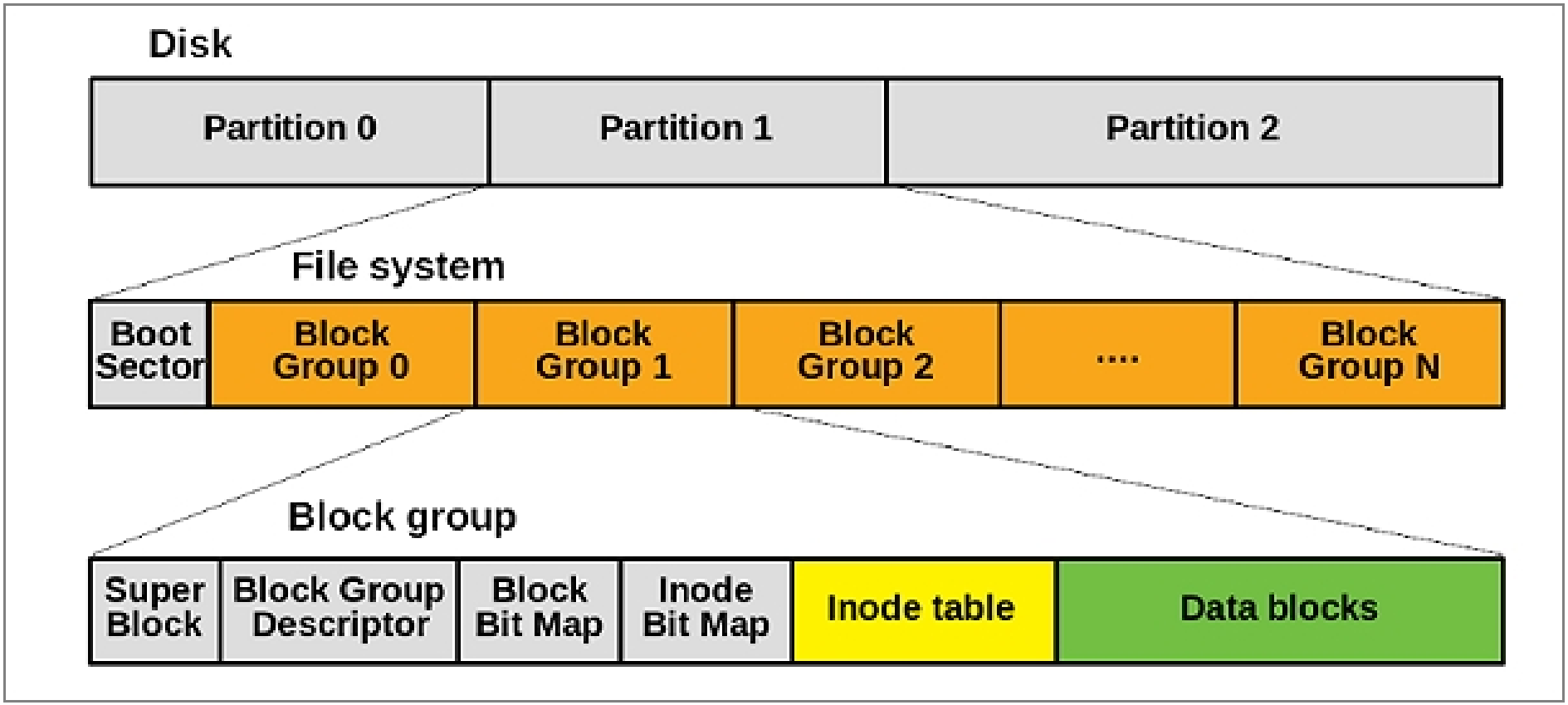


Figure: The filesystem is divided into several block group (inspired by FFS)

Superblock

Robustness

Robustness. Many features designed for fsck.
How to check, when to check and what if check fails?

- last_check
- check_interval
- mount_count
- max_mount_count
- state
- errors_behavior

Duplicated superblock and group descriptor tables. (Every block group has one, except `sparse_super` is set)
Read https://github.com/Tiger1218/imoutOS/blob/devel-ext2-low-layer-implementation/fs/ext2/ext2.h for details.

Superblock

Extensibility

ext2 -> ext3 -> ext4: backward-compatible
Superblock:

- s_feature_compat
- s_feature_incompat
- s_feature_ro_compat

Enough free to add/delete properties.
Example: HTree, xattr, journal (even you can directly use ext2 driver to read a ext3 filesystem)

Block Group Descriptor Table

- block bitmap
- inode bitmap
- inode table
- free inodes count
- free blocks count
- used dir count (Why here?)

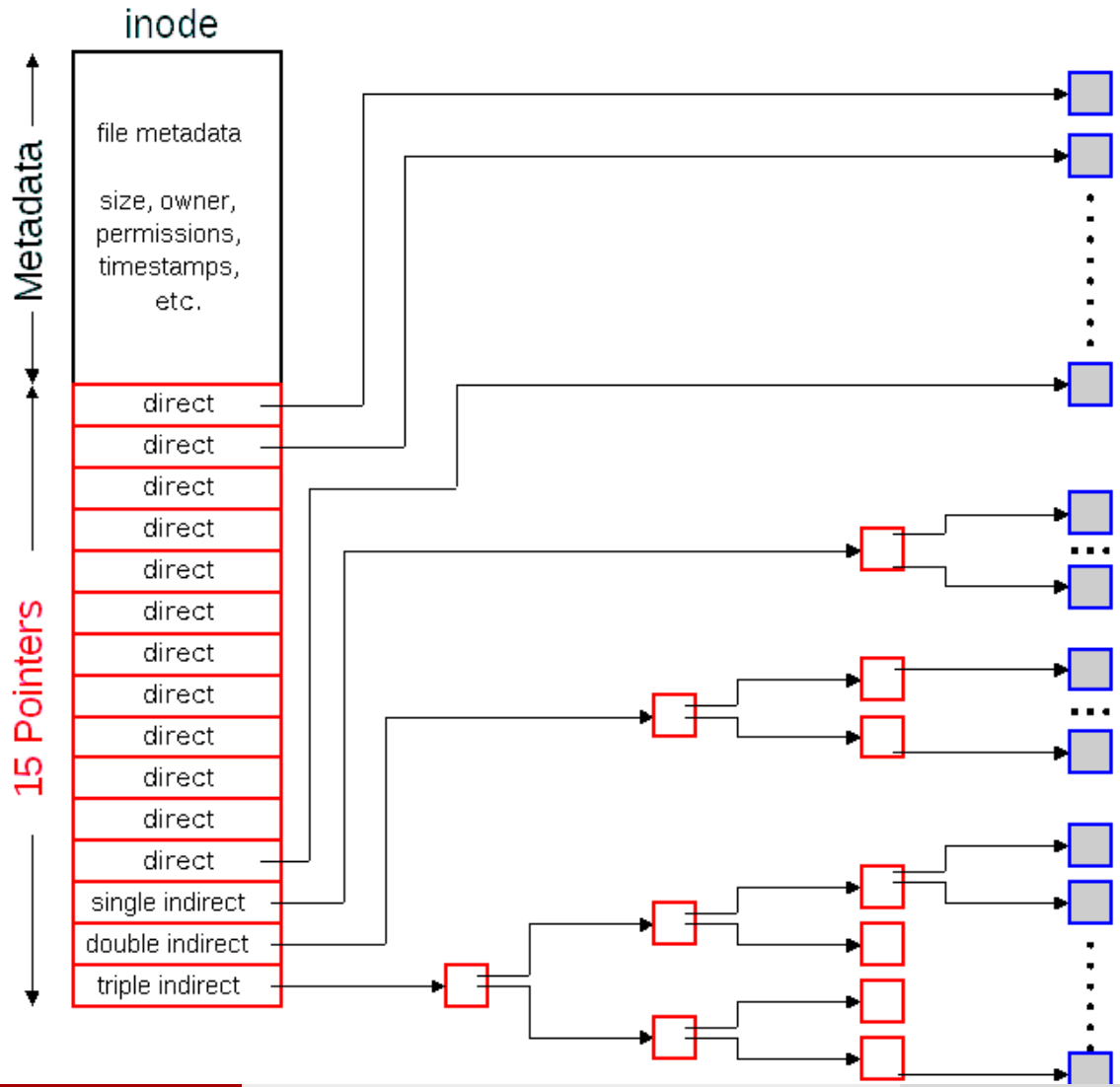
Also duplicated, but it requires a lot of blocks for storing it.

inode

- mode (UGOxRWX and filetype)
- a/c/m/d time (opened, change metadata, modify content, delete)
- uid/gid
- link_count
- size by block
- flags
- xattr blocks
- block (12 x direct, 1 x 2-level indirect, 1 x 3-level indirect, 1 x 4-level indirect)

inode

Explaination by Picture



dentry

Linked List

```
struct dir_entry {
    uint32_t    inode_num;
    uint16_t    record_len;
    uint8_t     filename_len;
    uint8_t     file_type;
    char        filename[EXT2_FILENAME_LEN];
} __packed;
```

Enable `dir_index` feature on ext2 filesystem to enable HTree Index instead of linked list. Become default on ext2 ver0+, and ext3.

Abstraction

Think before code.

```
struct block_group_descriptor * bgdget(uint32_t bgnum){
    uint32_t block_size = 1024 << sb_ext2.log_block_size;
    uint32_t bgs_per_block = block_size / sizeof(struct block_
    uint32_t blockid_to_read = bgnum / bgs_per_block;
    struct block_group_descriptor * bgt = malloc(block_size);
    read_from_block(blockid_to_read, bgt);
    return bgt[bgnum % bgs_per_block];
}
```

Use abstract to simplify the code instead of using procedural statement to damage the readability.

Decouple

```
read_from_bytes(1024, (char *)&sb_ext2,
                sizeof(struct super_block));
```

Superblock is different from **the superblock**. You never wanna use `1024 << sb_ext2.log_block_size` all the time.

Decouple

Only with the abstraction we can layer the problem.
Case study #1 : cache.

- Should I cache the inode/dentry, when and what size?
- Should I cache the block groups' inode table and bitmap, when?
- Should I cache data blocks / files, when and what size?
- Should I cache the r/wblock result, when?

Case study #2 : Get the inode

- How to deal with the block group?
- Should I turn `check_inode` from bitmap a function?
- If I want to cache the data within, how should we build a connection between it and inodes?

Decouple

Solution

- Each objects having abstraction should be cached.
- The cache process and invalidate process and whatever to keep the consistency should be implemented as a set of functions specify to the abstraction.
- So should the functional functions. And to keep codes clean, assign files with the abstractions.
- Properly handle the relations between the objects.

Congrats! You re-invent the OOP (Object Oriented Programming). You may want to check the Virtual Filesystem Switch mechanism for a detailed explaination.

References

M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A fast file system for unix," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 3, pp. 181–197, 1984.

R. Card, "Design and implementation of the second extended filesystem," in *Proc. First Dutch International Symposium on Linux, Dec. 1994*, 1994.

M. Cao, T. Y. Tso, B. Pulavarty, S. Bhattacharya, A. Dilger, and A. Tomas, "State of the art: Where we are with the ext3 filesystem," in *Proceedings of the Ottawa Linux Symposium (OLS)*, pp. 69–96, Citeseer, 2005.

Thank You!