【OS】Day47

Class	Operating System: Three Easy Pieces
Date	@March 8, 2022

【Ch40】File System Implementation

40.1 The Way to Think

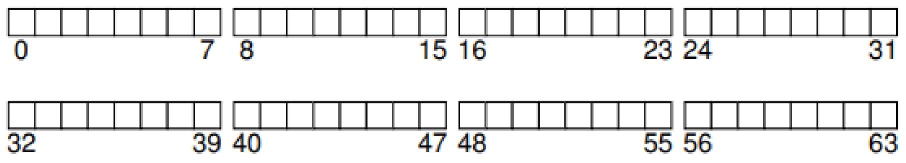
There are two different aspects of file systems:

1. The first is the data structures of the file system. In other words, what types of on-disk structures are utilized by the file system to organize its data and metadata?
2. The second aspect is its access methods. How does it map the calls made by a process such as `open()`, `read()`, `write()`, and etc.?

40.2 Overall Organization

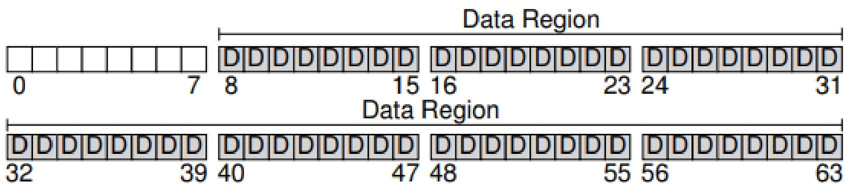
The first thing we need to do is to divide disk into blocks. Simple file systems use just one block size, and let's choose a commonly-used size of 4KB.

Assume we have a really small disk, with just 64 blocks:



What should we store to build a file system?

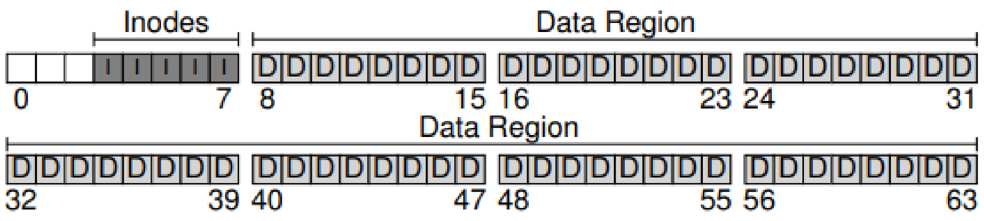
The first thing that comes up to mind is user data. In fact, most of the space in any file system is user data. Let's call the region of the disk we use for user data the data region, and reserve a fixed portion of the disk for these blocks. Say the last 56 of 64 blocks on the disk.



The file system has to track information about each file. This information is a key piece of metadata, and tracks things like which data blocks(in the data region) comprise a file, the size of the file, its owner and access rights, access and modify times, and other similar kinds of information.

To store this information, file systems usually have a structure called an inode.

To accommodate inodes, we'll need to reserve some space on the disk for them. Let's call this portion of the disk the inode table. It uses 5 of our 64 blocks of inodes:



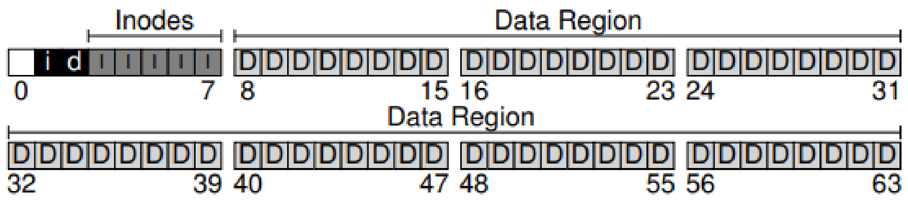
The inodes are typically not that big, for example 128 or 256 bytes. Assume 256 bytes per inode, a 4-KB block can hold 16 inodes. Our file system contains 80 total indoes, and this number represents the maximum number of files we can have in our system.

One primary component that is still needed is some way to track whether inodes or data blocks are free or allocated. Such allocation structures are thus a requisite element in any file system.

Many allocation-tracking methods are possible. For example, we could use a free list that points to the first free block, which then points to the next free block.

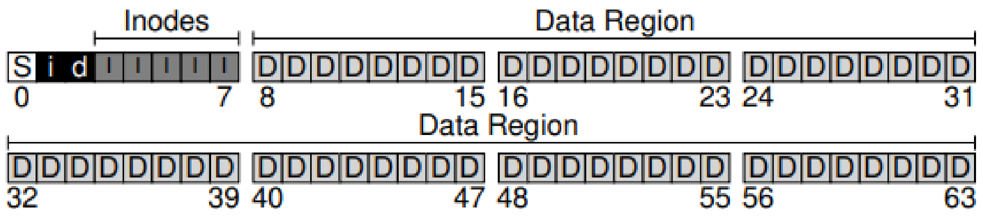
We instead choose a simple and popular structure known as a bitmap, one for the data region(the data bitmap), and one for the inode table(the inode bitmap).

A bitmap is a simple structure: each bit is used to indicated whether the corresponding object/block is free(0) or in-use(1). And thus our new on-disk layout, with an inode bitmap and a data bitmap:



This is a bit of overkill to use an entire 4-KB block for these bitmaps. Such a bitmap can track whether 32K objects are allocated. However, we just use an entire 4-KB block for each of these bitmaps for simplicity.

We reserve the last block for the superblock, denoted by an S in the diagram below. The superblock contains information about this particular file system, including how many incodes and data blocks are in the file system(80 and 56 in this case), where the inode table begins(block 3), and so forth. It will also include a number to identify the file system type.


Inodes
Data Region
S
i
d
I
D
0
7
8
15
16
23
24
31
Data Region
32
39
40
47
48
55
56
63