【OS】Day47(2)

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

【Ch40】File System Implementation(2)

40.3 File Organization: The Inode

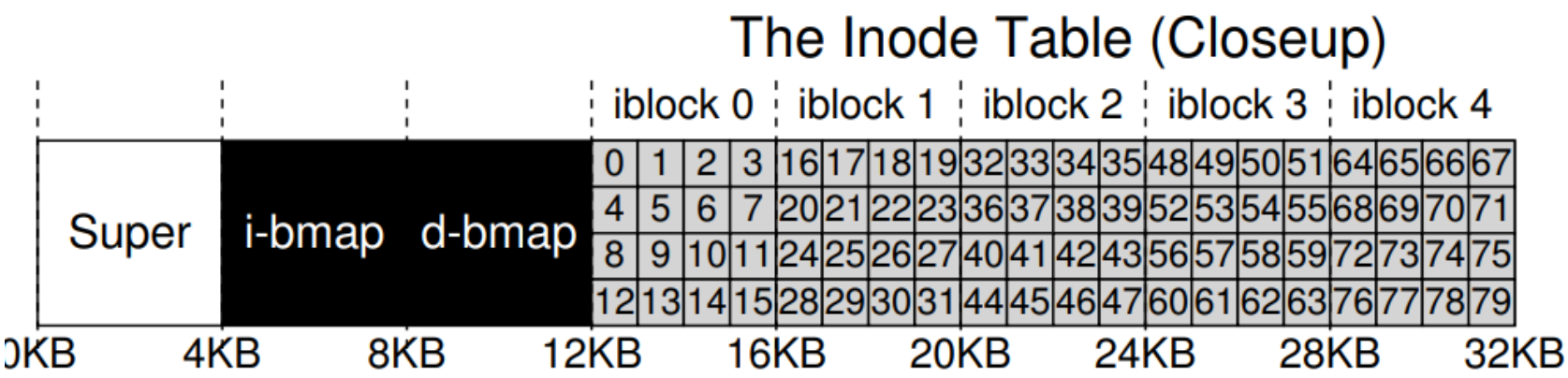
One of the most important on-disk structures of a file system is the inode. The name inode is short for index node. The name is used because these nodes were originally arranged in an array, and the array indexed into when accessing a particular node.

Each inode is implicitly referred to by a number(called the i-number), which we've earlier called the low-level name of the file.

In vsfs, given an i-number, we should directly be able to calculate where on the disk the corresponding inode is located.

For example, take the inode table of vsfs as above: 20KB in size(five 4KB blocks) and thus consisting of 80 indoes. Further assume that the inode region starts at 12KB(i.e. the superblock starts at 0KB, the indoe bitmap is at 4KB, the data bitmap at 8KB).

In vsfs, we thus have the corresponding structure:



To read indoe number 32, the file system would first calculate the offset into the inode region($32 \times sizeof(inode)$ or 8192), add it to the start address of the inode table on disk(`inodeStartAddr` = 12KB), and thus arrive upon the correct byte address of the desired block of indoes: 20KB.

Recall that disks are not byte addressable, but rather consists of a large number of addressable sectors, usually 512 bytes. Thus, to fetch the block of inodes that contains inode 32, the file system would issue a read to sector $\frac{20 \times 1024}{512}$, or 40, to fetch the desired indoe block.

The sector address `sector` of the inode block can be calculated as follows:

```
blk = (inumber * sizeof(inode_t)) / blockSize;
sector = ((blk * blockSize) + inodeStartAddr) / sectorSize;
```

Inside each inode is virtually all of the information we need about a file: its type, its size, the number of blocks allocated to it, protection information, some time information, including when the file was created, modified, or last accessed, as well as information about where its data blocks reside on disk.

We refer to all such information about a file as metadata.

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists

Figure 40.1: **Simplified Ext2 Inode**

The Multi-Level Index

To support bigger files, file system designers have had to introduce different structures within inodes. One common idea is to have a special pointer known as an indirect pointer.

Instead of pointing to a block that contains user data, it points to a block that contains more pointers, each of which point to user data. Thus, an inode may have some fixed number of direct pointers and a single indirect pointer.

If a file grows large enough, an indirect block is allocated (from the data-block region of the disk), and the inode's slot for an indirect pointer is set to point to it.

We might want to support even larger files. To do so, just add another pointer to the inode: the double indirect pointer. This pointer points to a block that contains pointers to indirect blocks, each of which contain pointers to data blocks.