Lecture 20: File System Implementation

Operating Systems

Content taken from: https://pages.cs.wisc.edu/~remzi/OSTEP/

https://www.cse.iitb.ac.in/~mythili/os/

Last Lecture

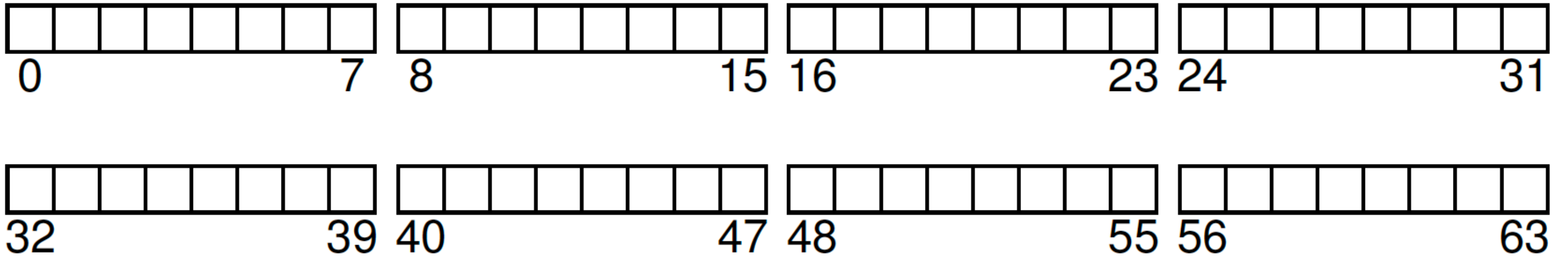
- System calls related to creating and accessing files
  - File Creation
  - File Reading and Writing
    - Sequential and random
- Directory-related system calls

File System

- An organization of files and directories on disk
- OS has one or more file systems
- Two main aspects of file systems
  - Data structures to organize data and metadata on disk
  - Implementation of system calls like open, read, write using the data structures
- We will study **vsfs** (Very Simple File System) – a simplified version of UNIX file system

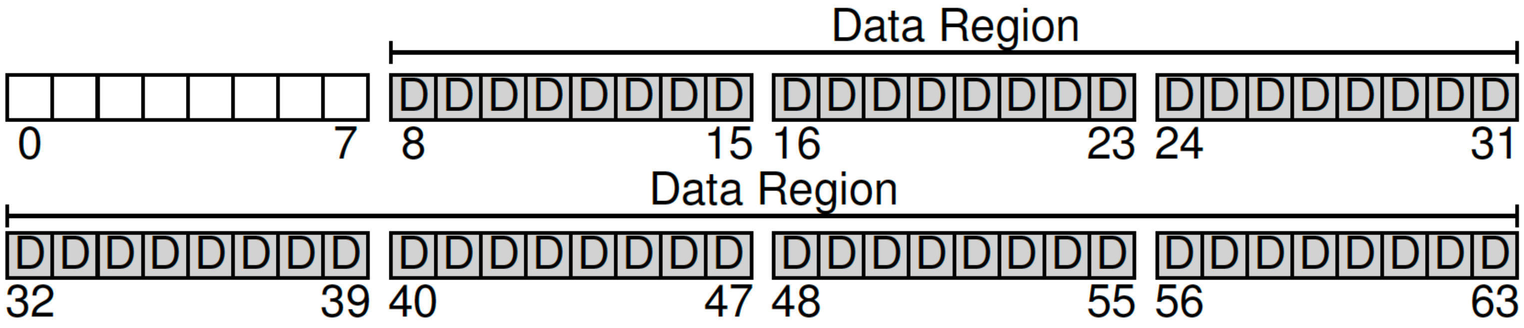
Overall Organization

- The entire disk can be thought of as divided into blocks
- So, disk is nothing but a series of blocks
  - Each block is of size 4 KB (commonly-used size)
  - Blocks are addressed from 0 to N-1
- File system organizes files onto blocks
  - System calls translated into reads and writes on blocks
- In the below example, we have a small disk having 64 blocks



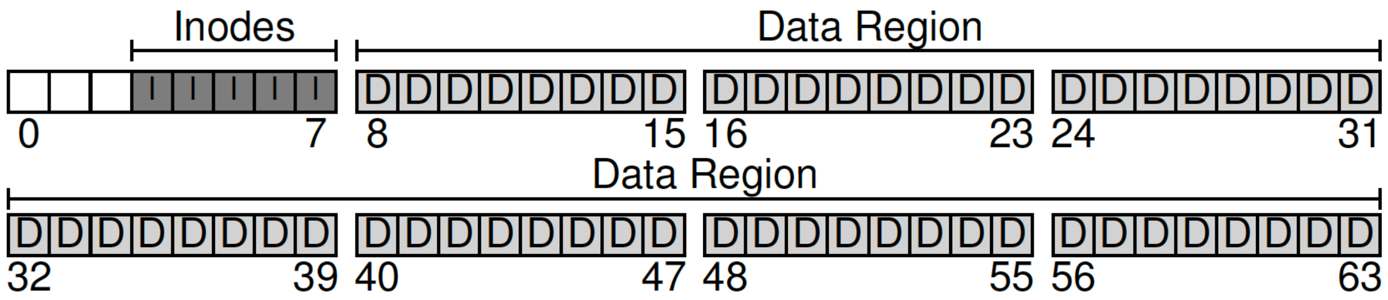
Data Region

- Region of the disk we use for storing user data



Inode Table

- Region of the disk which tracks information (metadata) about each file
- Each file/directory has one inode associated with it
- **Metadata**
  - Which data blocks comprise a file?
  - File Size
  - Owner and Access rights
  - Access time, modify time
  - Etc.



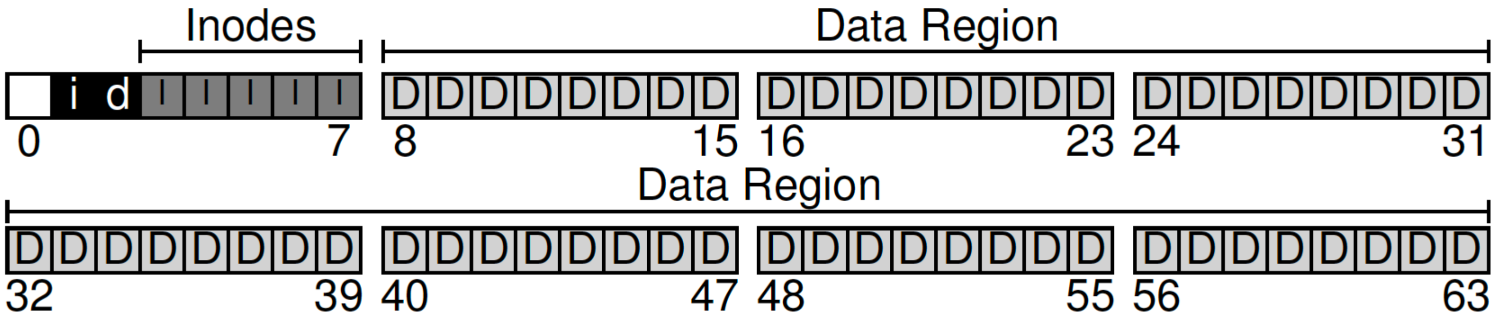
- **Example**
  - Size of one inode = 256 bytes
  - One block (4KB) can hold 16 inodes.
  - Maximum number of files which can be stored in the above file system = 80

Allocation structures

- Track whether inodes or data blocks are free or allocated
- Many options for these structures:
  - **Free list** (linked list pointing to the first free block)
  - **Bitmap**

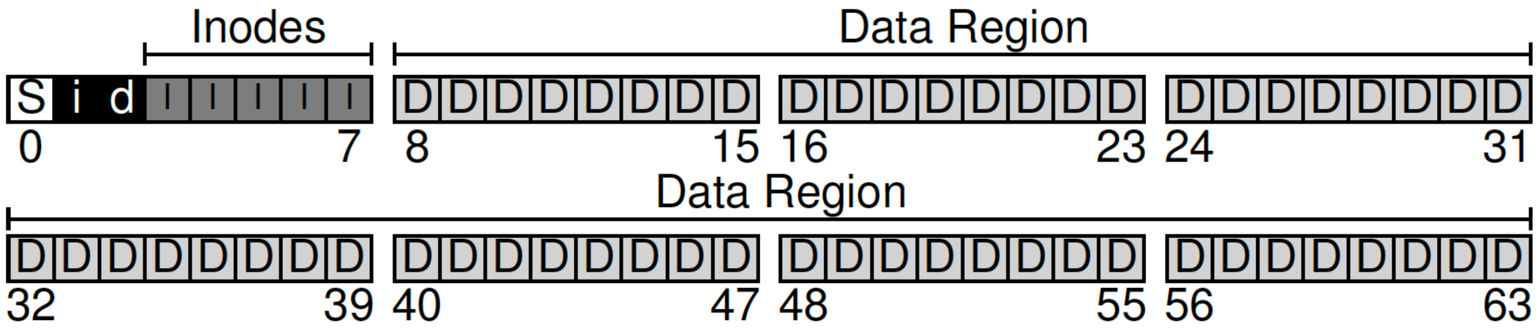
Bitmap for tracking free inodes or blocks

- One bitmap for tracking inode
- Another bitmap for tracking data blocks
- Each bit in the bitmap is used to indicate whether the corresponding inode or data block is free (0) or in-use (1)



Superblock

- Contains meta data about the disk organization
  - Number of inodes
  - Number of data blocks
  - From which block the inode table begins?
  - From which block the data region begins?
  - Etc.



Inode

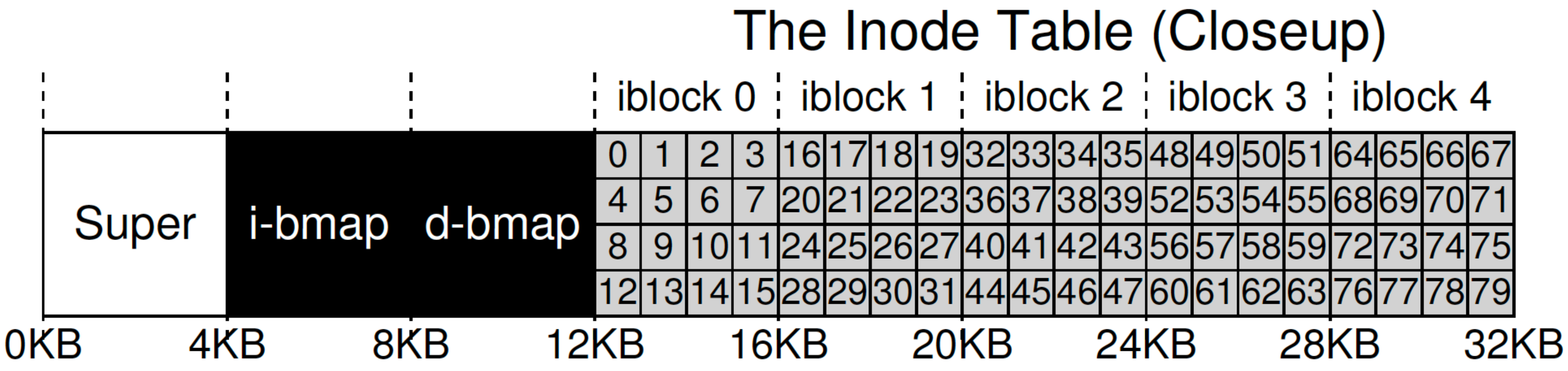
- Inode is short for **index node**
- Inode is implicitly referred to by a number
  - **inode number** or **i-number** [os-level identifier we talked about earlier]
- Each inode contains all the **metadata** about the file
- Also, stores information about the disk blocks where the actual file contents are stored

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**

Given an i-number, which disk sector do we need to read for fetching this inode?

- Inode no. = 32
- Size of inode = 256 B
- Block size = 4KB
- Sector size = 512 B
- Inode Start Address
  - 12 KB

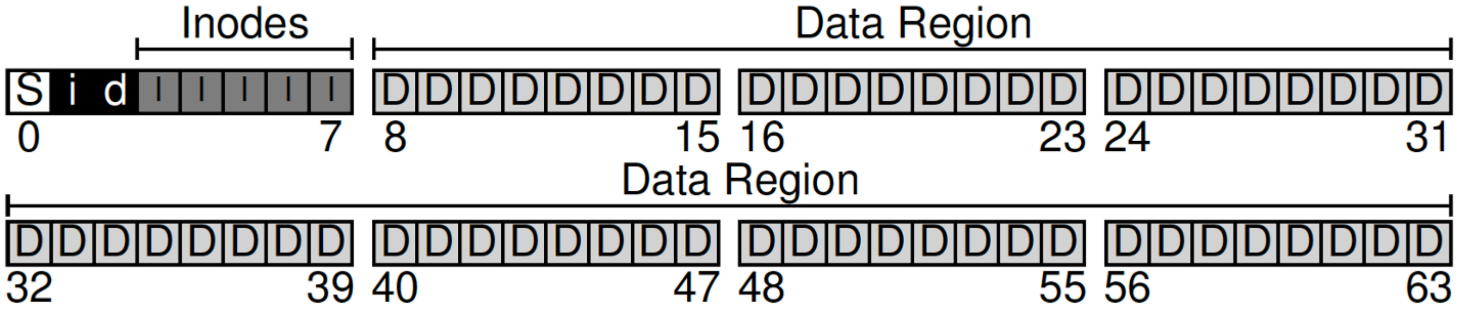



```
blk    = (inumber * sizeof(inode_t)) / blockSize;
sector = ((blk * blockSize) + inodeStartAddr) / sectorSize;
```

- Inode 32’s relative position = 32*256 = 8192 = 8KB
- blk (iblock number) = 8k/4k = 2
- Sector = (8k + 12k)/512 = 40

How does an inode refer to data blocks?

- **Direct pointers**
  - Store an array of disk addresses (direct pointers) inside the inode
  - Each disk address (pointer) refers to one disk block that belongs to the file
- **What is the problem in this approach?**
  - What happens if the file size is very big?



Multi-Level Index

- Store Indirect pointer(s) inside an inode
- An **indirect pointer** points to a block that contains more pointers, each of which point to user data block.
- **Double indirect pointer**
  - Points to a block that contains pointers to indirect blocks (containing indirect pointers).
  - Each pointer in an indirect block contains pointers to data blocks
- **Triple indirect pointer**

Examples

- How large a file can we store using an inode which has 12 direct pointers and one indirect pointer?
  - Block size = 4 KB
  - Pointer size (disk address size) = 4 bytes

- 4K blocks and 4 byte disk addresses leads to 1024 pointers
- The file size is (12 + 1024) * 4K = 4144 KB

Examples

- How large a file can we store using an inode which has 12 direct pointers, one indirect pointer and one double indirect pointer?
  - Block size = 4 KB
  - Pointer size (disk address size) = 4 bytes

- 4K blocks and 4 byte disk addresses leads to 1024 pointers
- With double indirection, each of the 1024 pointers, point to another 1024 pointer blocks
- The file size is (12 + 1024 + 1024^2) * 4K = 4 GB

Directory Organization

- A directory contains a list of **(entry name, inode number)** pairs

```
inum | reclen | strlen | name
  5      12        2     .
  2      12        3     ..
 12      12        4     foo
 13      12        4     bar
 24      36       28     foobar_is_a_pretty_longname
```

- Why do we have record length and string length separately?
  - Deleting a file can leave an empty space (entry) in the middle of the directory
  - A new entry may reuse such a “deleted” entry and thus have extra space within

Free Space Management

- How does a file system track which inodes and data blocks are free?
- In vsfs, we have two bitmaps for tracking free blocks:
  - Inode bitmap
  - Data block bitmap
- During file creation, file system will search through these bitmaps to find free inodes and data blocks.

Reading a File from Disk

- Consider reading a **12KB** file **/foo/bar** (3 blocks)

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data [0]	bar data [1]	bar data [2]
open(bar)			**read**							
						read				
				read						
							read			
					read					
read()					**read**					
								read		
					write					
read()					**read**					
									read	
					write					
read()					**read**					
										read
					write					

Figure 40.3: **File Read Timeline (Time Increasing Downward)**

Writing a File to Disk

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data [0]	bar data [1]	bar data [2]
create (/foo/bar)			read							
						read				
				read						
							read			
		read								
		write								
							write			
					read					
					write					
				write						
write()					read					
	read									
	write									
								write		
					write					
write()					read					
	read									
	write									
									write	
					write					
write()					read					
	read									
	write									
										write
					write					

Figure 40.4: **File Creation Timeline (Time Increasing Downward)**

Caching

- **Cache important blocks in memory**
  - **Static partitioning:** Allocate a fixed size of memory for caching file system blocks
  - **Dynamic partitioning:** Allocate memory more flexibly across virtual memory and file system
- **Write Buffering:** Delaying/Buffering writes
  - **Batch** some writes into a smaller set of I/Os
  - **Schedule** the set of I/Os more intelligently
  - **Avoid** some writes