File System Implementation pt. 1

Operating Systems
Based on: Three Easy Pieces by Arpaci-Dusseaux

Moshe Sulamy

Tel-Aviv Academic College

Very Simple File System

- File system: pure software
  - Many different file systems exist
- Start with a cast study: **vsfs**
  - Simplified version of typical UNIX file system

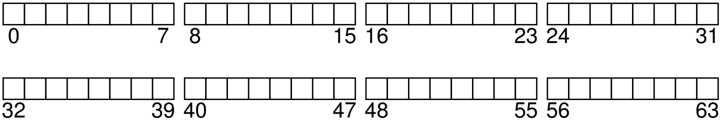
How can we build a simple file system?

Two Aspects

- **Data structures**
  - What type of on-disk structures?
- **Access methods**
  - How to map calls (`open()`, `read()`, `write()`, etc.)?
  - Read which structures during which calls?

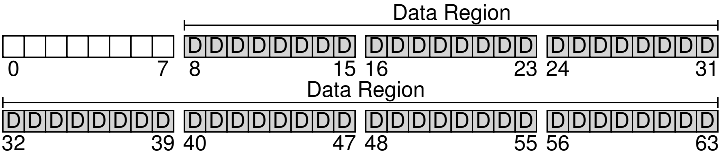
Overall Organization

- Divide disk into **blocks**
  - Addressed 0 to $N-1$
  - Commonly-used size: 4 KB



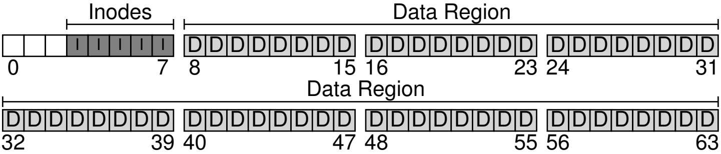
Overall Organization

- Reserve **data region** for user data
  - e.g., fixed portion: 54 of 64 blocks


Data Region
D D D D D D D D D D D D D D D D D D D D D D D D
0
7
8
15
16
23
24
31
Data Region
D D D D D D D D D D D D D D D D D D D D D D D D D D D D D D D D
32
39
40
47
48
55
56
63

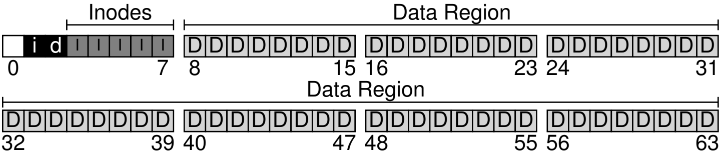
Overall Organization

- **Metadata** on file
  - Data blocks, size, owner, access writes, times, etc.
  - Usually in **inode** structure
  - ~256 bytes per inode



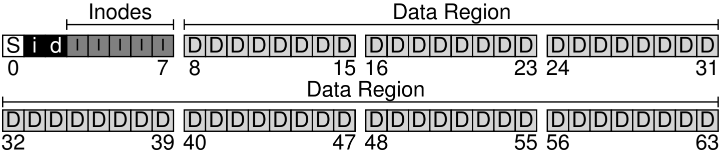
Overall Organization

- **Allocation structure**
  - Track whether inodes or data blocks are free or allocated
  - **Free list**
  - **Bitmap**
    - One for data region and one for inode table
    - Each bit indicates free (0) or in-use (1)



Overall Organization

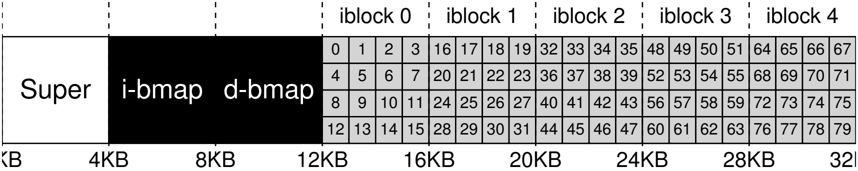
- Remaining block: **superblock**
  - Information about this file system
  - e.g., number of inodes and data blocks, where inode table begins, magic number to identify file system type, etc.
- On mount, OS reads superblock first to initialize



The Inode

- **Index node**
  - Referred to by a number: **i-number** (**low-level name**)
  - Example: 20KB inode table → 80 inodes
  - Read inode 32:

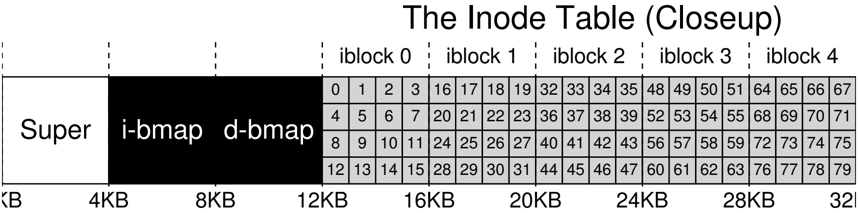
The Inode Table (Closeup)



iblock 0				iblock 1				iblock 2				iblock 3				iblock 4			
0	1	2	3	16	17	18	19	32	33	34	35	48	49	50	51	64	65	66	67
4	5	6	7	20	21	22	23	36	37	38	39	52	53	54	55	68	69	70	71
8	9	10	11	24	25	26	27	40	41	42	43	56	57	58	59	72	73	74	75
12	13	14	15	28	29	30	31	44	45	46	47	60	61	62	63	76	77	78	79

The Inode

- **Index node**
  - Referred to by a number: **i-number** (**low-level name**)
  - Example: 20KB inode table → 80 inodes
  - Read inode 32: $32 \cdot sizeof(inode) + startAddr = 8KB + 12KB$



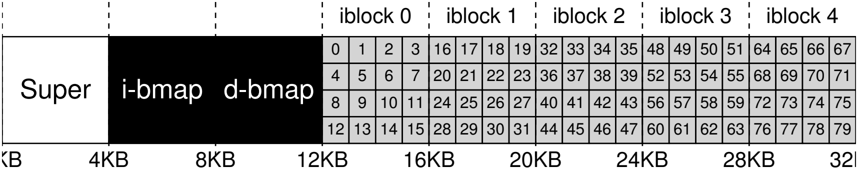
The Inode Table (Closeup)

iblock 0				iblock 1				iblock 2				iblock 3				iblock 4			
0	1	2	3	16	17	18	19	32	33	34	35	48	49	50	51	64	65	66	67
4	5	6	7	20	21	22	23	36	37	38	39	52	53	54	55	68	69	70	71
8	9	10	11	24	25	26	27	40	41	42	43	56	57	58	59	72	73	74	75
12	13	14	15	28	29	30	31	44	45	46	47	60	61	62	63	76	77	78	79

The Inode

- **Index node**
  - Referred to by a number: **i-number** (**low-level name**)
  - Example: 20KB inode table → 80 inodes
  - Read inode 32: $32 \cdot \mathit{sizeof}(\mathit{inode}) + \mathit{startAddr} = 8KB + 12KB$
  - Read disk sector $\frac{20\times 1024}{512} = 40$

The Inode Table (Closeup)



iblock 0				iblock 1				iblock 2				iblock 3				iblock 4			
0	1	2	3	16	17	18	19	32	33	34	35	48	49	50	51	64	65	66	67
4	5	6	7	20	21	22	23	36	37	38	39	52	53	54	55	68	69	70	71
8	9	10	11	24	25	26	27	40	41	42	43	56	57	58	59	72	73	74	75
12	13	14	15	28	29	30	31	44	45	46	47	60	61	62	63	76	77	78	79

The Inode

- Where are the data blocks?
  - One or more **direct pointers**
  - Each pointer refers to one disk block
  - Limited: no support for big files

Multi-Level Index

- **Indirect pointers**
    - Point to a block that contains more pointers
    - Each points to user data
    - Inode: fixed number of direct pointers, single indirect pointer
- File grows large: allocate indirect block
    - Point inode's indirect pointer to it

Multi-Level Index

- Multi-level index approach:
  - Double indirect pointer: points to block of indirect pointers
  - Triple indirect pointer: points to block of double indirect pointers
- Example:
  - Block size 4KB, 4-byte pointers
  - 12 direct pointers, both single and double indirect block

Multi-Level Index

- Multi-level index approach:
  - Double indirect pointer: points to block of indirect pointers
  - Triple indirect pointer: points to block of double indirect pointers
- Example:
  - Block size 4KB, 4-byte pointers
  - 12 direct pointers, both single and double indirect block
  - Can accommodate 4GB file $((12 + 1024 + 1024^2) \times 4KB)$

Multi-Level Index

- Many file systems use multi-level index
  - Linux ext2 and ext3, NetApp's WAFL, UNIX file system
  - SGI XFS and Linux ext4 use **extents**
- **Extents**
  - Disk pointer plus length
  - Avoids large metadata per file (pointer for every block)

Multi-Level Index

- Measurement summary:

Most files are small	$\sim$2K most common size
Average file size growing	Almost 200K
Most bytes are stored in large files	Few big files use most of space
File systems contain lost of files	Almost 100K on average
File systems are roughly half full	Even as disks grow
Directories are typically small	Most have 20 or fewer entries

Linked-Based Approaches

- Use **linked list**
  - One pointer inside inode → first block of file
  - End of data block → another pointer
- Performs poorly for some allocations
  - e.g., read last block of file, random access
  - Solution? instead of next pointers, in-memory table of links
- Used by **FAT** (**file allocation table**) file system
  - Directory entries instead of inodes (hard links impossible)
  - Classic Windows file system before **NTFS**