

5118020-03 Operating System

File System Implementations

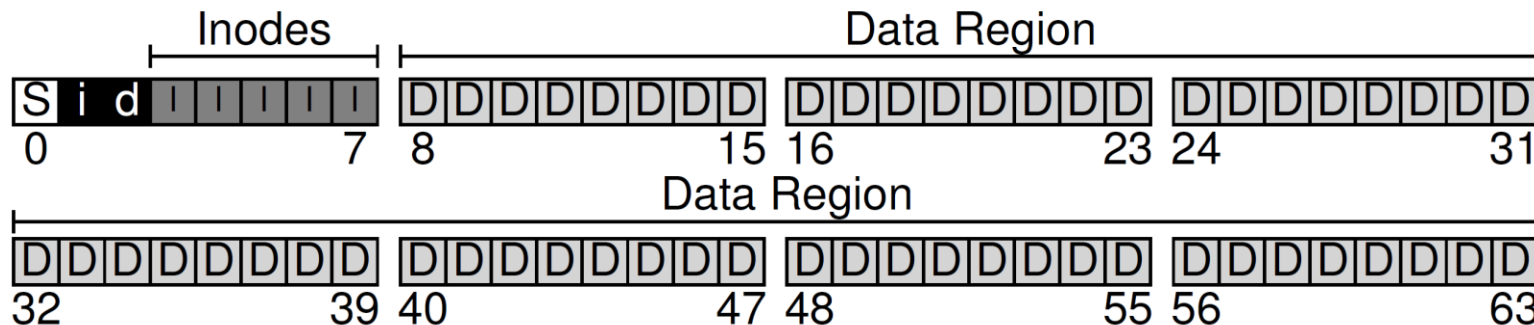
Chapters 40 and 41

Shin Hong

Very Simple File System

2

- A simplified imitation of the UNIX file systems to build a mental model to understand the data structure and the operations of the file systems
- On-disk organization
 - a disk partition consists of an array of N blocks, each of size 4 KB
 - four regions: **data region** to hold user data, **inode table** to hold metadata, **inode bitmaps**, and **superblock**
- Example
 - 64 blocks where 5 blocks are for inode and 56 for data region
 - 256 bytes per inode, thus, total 80 inodes can be accommodated
 - one block for data bitmap, one for inode bitmap and one for superblock



File System
Implementation

5118020-03
Operating System

2024-06-10

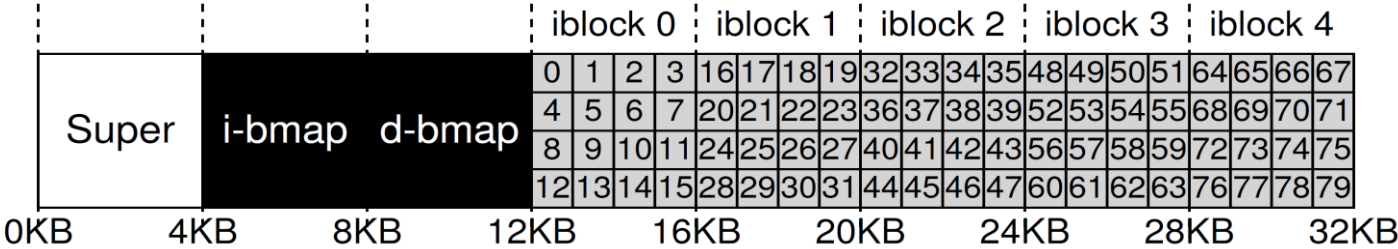
Inode (1/2)

- Each inode (index node) is referred to by a unique number (identifier)
- The metadata of a file is written on a single inode

- Ex. inode structure for ext-2

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

- ex. inode table layout



Inode (2/2)

4

- Indirect pointer
 - To accommodate a large file, an inode may use one data block to hold a set of direct pointers each of which points to a data block
 - Ex. 1-level indirect pointer
 - an inode have 12 direct pointers and 1 indirect pointer
 - for 4-byte block addresses, one block can hold 1024 direct pointers through which a file can grow upto 1036×4 KB size
 - Ex. 2-level indirect pointer (e.g., ext2, ext3)
 - the indirect pointer of an inode points to a block of indirect pointers each of which points to another block of direct pointers
 - for 4-byte block address, a file can grow upto $1024 \times 1024 \times 4$ KB size

File System
Implementation

5118020-03
Operating System

2024-06-10

Directory Organization

5

- A directory is a list of entry name-inode number pairs
 - a directory is treated as a special type of file, thus the content resides in the data blocks
 - example

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

File System
Implementation

5118020-03
Operating System

2024-06-10

Free Space Management

6

- vsfs tracks free inodes using inode bitmap and data bitmap
- operations involved in a file creation
 1. search for a free inode in the inode bitmap, and allocate it to the file
 2. mark the corresponding inode bit (in the inode bitmap) as used
 3. search for a free data block in the data bitmap, and allocate it to the file
 4. mark the corresponding data bit (in the data bitmap) as used
- Some OS's pre-allocate a sequence of free blocks (e.g., 8) when a file is created.
 - to guarantee a portion of the file to be contiguously aligned

File System
Implementation

5118020-03
Operating System
2024-06-10

Access Path (1/2)

- Assumption
 - the superblock was already on the main memory when the file system is mounted
 - the other blocks are all placed on the disk
- Reading a file from a disk
 - Ex. `open("/foo/bar", O_RDONLY)`
 - find the inode for '/' by checking the superblock attributes (usually, 2)
 - read the inode for '/'; read the inode and iterate over the list to find a direntry for 'foo'; find the inode for 'foo'
 - read the inode for 'foo' to find the inode for 'bar'
 - read the inode of 'bar'; check that all permissions are alright for the user.
 - allocate a file descriptor in the open file table and returns the descriptor

	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()					write					
read()					read				read	
read()					write					
					read					read
					write					

Access Path (2/2)

- Writing to a file
 1. open a file
 - create a file if the file does not exist
 2. write data
 - allocate a new data block if it is not overwriting
 - update the file size at the inode
- Ex. writing to `/foo/bar`
 - 10 block accesses for a file creation
 - 5 block accesses for each block allocation

	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 write	read	read		read	read	write		
write()	read write				read write					
write()	read write				write read					write
write()	read write				write read					write

Caching and Buffering

9

- Reading/writing files can be expensive for frequent disk block accesses
- File systems use main memory to cache important blocks
 - static partitioning: a fixed-size cache to hold popular blocks (with LRU)
 - dynamic partitioning: integrate virtual memory pages and file system pages into a unified page cache
- Write buffering
 - delay writing to the storage device, and update the buffered status
 - the file system can batch some updates into a small set of I/O operations
 - the file system can have better chance for disk scheduling

File System
Implementation

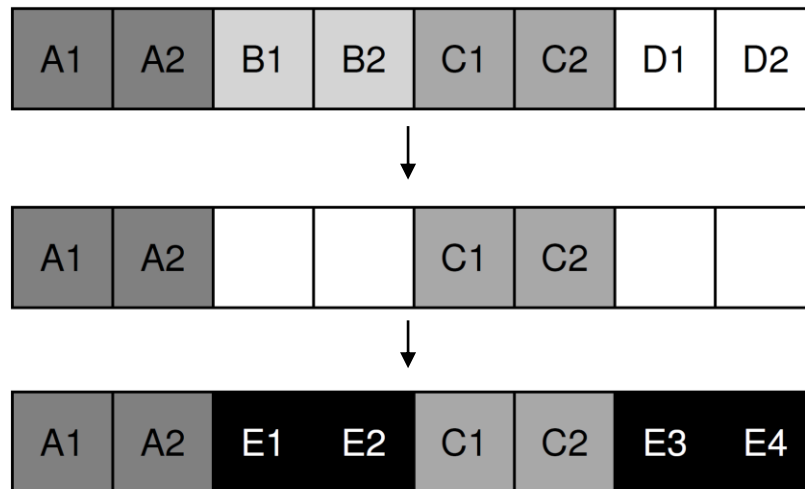
5118020-03
Operating System
2024-06-10

Problem of Very Simple File System

10

- Poor performance since it randomly accesses disk blocks
 - heavy seek costs for random accesses
 - careless free space management results in fragmentation

- example



- small disk block size incurs heavy positioning overhead

File System
Implementation

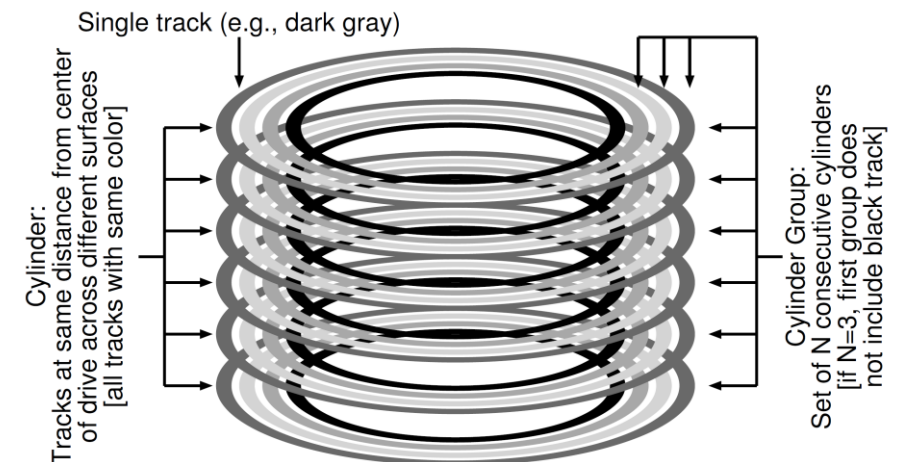
5118020-03
Operating System

2024-06-10

Fast File System (FFS)

11

- a file system structure and allocation policies consider the characteristics of disk device for performance optimization
- cylinder group
 - a cylinder is a set of tracks on different surfaces that are the same distance from the spindle
 - FFS aggregates N consecutive cylinders as a cylinder group
 - Accessing two files in a cylinder group will not result in long seek
 - FFS manages a storage device as a set of cylinder groups
 - allocate the related inode, datablocks, and other structures within the same cylinder group



Policies

12

- Collocate related data while keeping unrelated data apart
- Placement heuristics
 - put the inode and the data blocks of the same file (or the same directory) to the same group
 - place all files in a directory to the group where the directory data is placed, because they are often accessed together
 - example: /a/c, /a/d, /a/e, /b/f

group	inodes	data
0	/-----	/-----
1	acde-----	accddee---
2	bf-----	bff-----
3	-----	-----
4	-----	-----
5	-----	-----
6	-----	-----
7	-----	-----

group	inodes	data
0	/-----	/-----
1	a-----	a-----
2	b-----	b-----
3	c-----	cc-----
4	d-----	dd-----
5	e-----	ee-----
6	f-----	ff-----
7	-----	-----

File System
Implementation

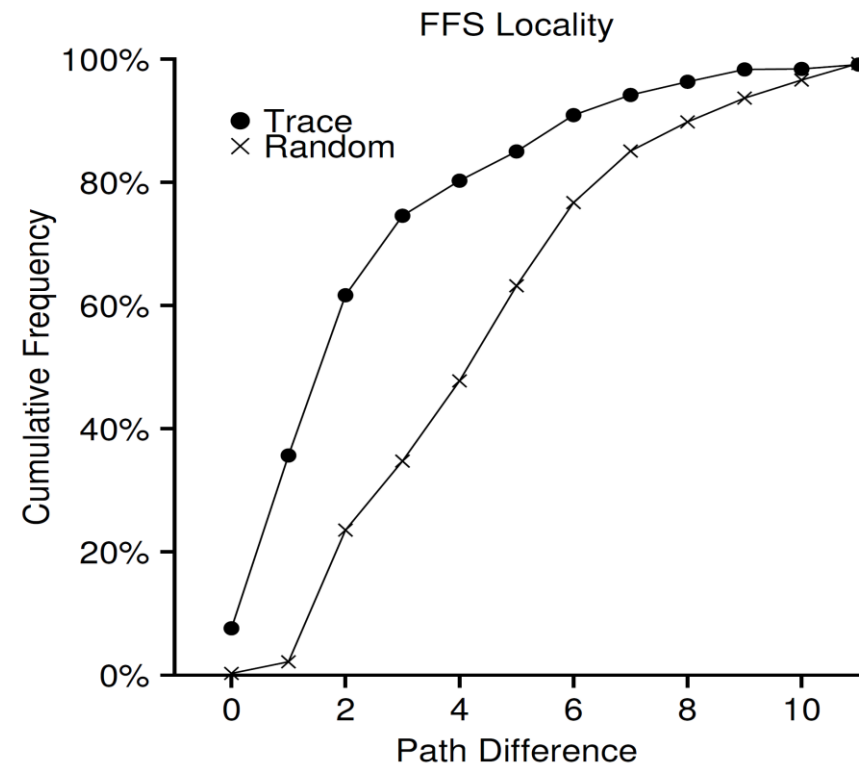
5118020-03
Operating System

2024-06-10

File Locality

13

- It is likely that the next file that a user accesses has a short path difference to the most recently accessed file
 - the path difference between two files is the maximum distance between the common ancestor and each file
 - 0 for the same file, 1 for the two files in the same dir
- The observation from the SEER trace shows that the path difference of more than 60% adjacent file accesses is less than equal to 2



File System
Implementation

5118020-03
Operating System

2024-06-10

Handling Large File Exceptionally (1/2)

- Without an exceptional handling, a large file will consume a block group and hurt the locality chance of the other files in the same block group

- E.g.

group	inodes	data
0	/a-----	/aaaaaaaaa aaaaaaaaaa aaaaaaaaaa a-----
1	-----	-----
2	-----	-----

- For an exceptionally large file, FFS spread the data block across multiple block groups to avoid the file to occupy a single block group

- E.g.

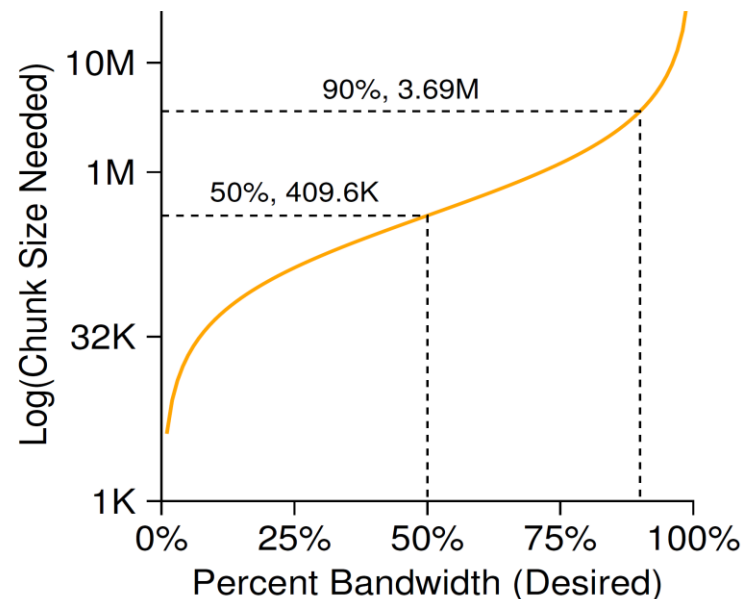
group	inodes	data
0	/a-----	/aaaaa-----
1	-----	aaaaa-----
2	-----	aaaaa-----
3	-----	aaaaa-----
4	-----	aaaaa-----
5	-----	aaaaa-----
6	-----	-----

Handling Large File Exceptionally (2/2)

15

- How much data should be placed in a block group
 - FFS approach
 - the first twelve (pointed by direct pointers) are placed in the same block group
 - each set of the data blocks pointed through an indirect pointer is placed on a different block group
 - more analytic approach: calculate the chunk size to provide an expected bandwidth
 - e.g., for a disk with 10 ms avg. positioning time and 40 MB/s transfer rate, to provide 50% bandwidth (a half time for seeking, a half time for transfer):

$$\frac{\text{Chunk Size}}{40 \text{ MB/sec}} = 10 \text{ ms}$$



File System
Implementation

5118020-03
Operating System

2024-06-10