IE411: Operating Systems

File System Implementation

File systems

- A file is some unit of persistent data that we can refer to by name
- A file system (FS) implements the file abstraction
  - open(), read(), write(), close()
- Basic functions
  - translates application requests into disk block requests
  - imposes access control on our data
  - ensures consistency in the presence of failures

Outline

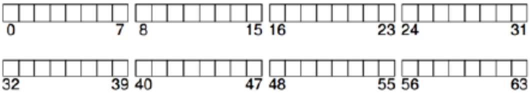
- What type of information does the file system need to manage to do its job?
- How does the file system organize data on a disk?
- How does the file system traverse this data when performing representative file system operations?

A Reference FS

- To build our file system we divide up the disk into a logical array of blocks (not the same as the disk sectors, which may be smaller!)
- Let's start with 4kB blocks (fairly common size)

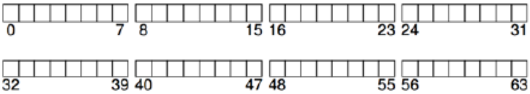
A Reference FS

- To build our file system we divide up the disk into a logical array of blocks (not the same as the disk sectors, which may be smaller!)
- Let's start with 4kB blocks (fairly common size)
- For example a small FS with 64 blocks



A Reference FS

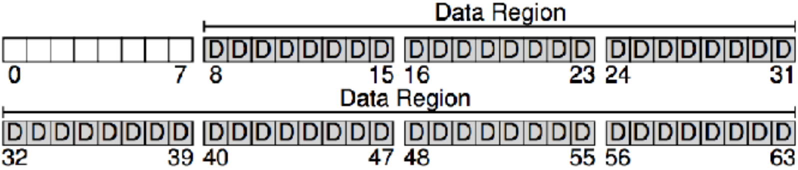
- To build our file system we divide up the disk into a logical array of blocks (not the same as the disk sectors, which may be smaller!)
- Let's start with 4kB blocks (fairly common size)
- For example a small FS with 64 blocks



- How big is the file system?

Data Region

- Let's reserve most of the file system for the users' data
- We will call that the Data Region

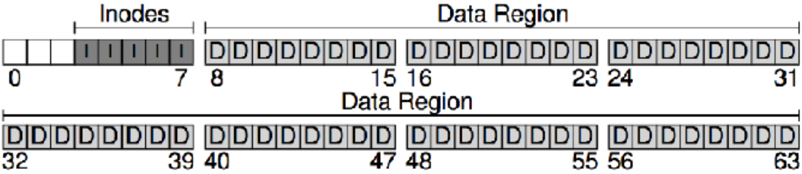


Inodes

- We need to keep track of locations of the file's contents, size, its creation date, etc (this is the metadata about the file)
- The structure that holds the metadata is called an inode
- Where do we store our inodes?

Inodes

- Inodes don't need a whole block (4kB) for storing the metadata for each file
- Assume 256 bytes should be good
- We will reserve 5 blocks for inodes



Question

We have 5 blocks for inodes, 4kB blocks, 256-byte inodes. How many files can we have on this file system?

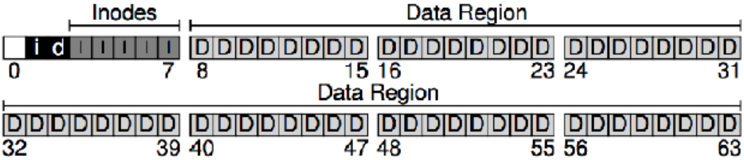
A. 5
B. 2^{12}
C. 2^{8}
D. 2^{20}
E. 80

Tracking allocations

- We need to track which inodes and which data blocks are free/used
- A bitmap is a simple structure to track such things

Tracking allocations

- We need to track which inodes and which data blocks are free/used
- A bitmap is a simple structure to track such things
- One bit per inode (80 bits), and one bit per data block (56 bits). But lets be lazy and use a whole block for each



Tracking allocations

- To set a single bitmap bit, you need to
  1. read an entire block into memory
  2. update that bit in memory
  3. write back the entire block

Managing metadata

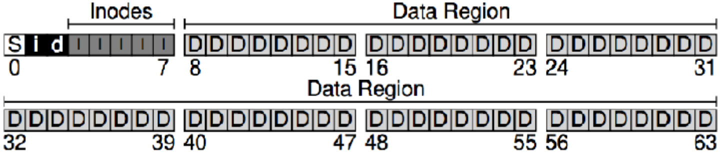
- How to we know where our structures are located?
- We need to know
  - bitmap locations
  - inode locations (inode table start)
  - number of inodes
  - data region start
  - number of data blocks

Superblock

- The superblock stores information about the whole file system

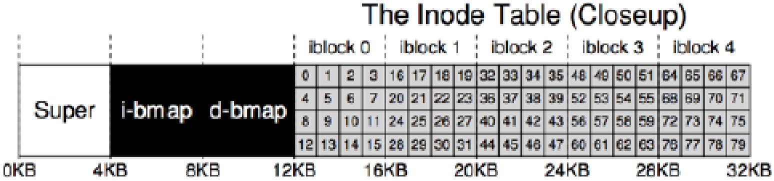
Superblock

- The superblock stores information about the whole file system
- Since the superblock is so important, it is often treated specially (stored at a known location)



Finding an inode

- Each inode on the system has a number
- To locate inode 32 you need to compute the address on disk:
  - $32 \star \text{sizeof(inode)} + \text{start of inode region} = 8\text{kB} + 12\text{kB} = 20\text{kB}$



Disk Sector

- Oops, hard disks are not byte addressable
- You have to read a whole sector, typically 512 bytes
- So what sector is address 20kB in?

Disk Sector

- Oops, hard disks are not byte addressable
- You have to read a whole sector, typically 512 bytes
- So what sector is address 20kB in?
  - (20*1024/512)=40

Referencing data blocks via pointers

- We need to know which blocks in the data region are associated with a specific file
- One option: have one or more direct pointers (disk addresses) inside the inode, each refers to one data block
- Since the inode has a fixed number of pointers, this fixes the max size of the file:
  - no of pointers $\star$ block size

Referencing data blocks via pointers

- We need to know which blocks in the data region are associated with a specific file
- One option: have one or more direct pointers (disk addresses) inside the inode, each refers to one data block
- Since the inode has a fixed number of pointers, this fixes the max size of the file:
    - no of pointers $\star$ block size
- Example: 12 direct pointers, 4kB block size
    - 48kB max file size

Referencing data blocks via pointers

- If we need bigger files we can use an additional indirect pointer which is a pointer to a data block, filled with pointers
- A 4kB block with 4 byte pointers = 1024 pointers in a block

Referencing data blocks via pointers

- If we need bigger files we can use an additional indirect pointer which is a pointer to a data block, filled with pointers
- A 4kB block with 4 byte pointers = 1024 pointers in a block
- If we have 12 direct pointers and 1 indirect pointer, what is the maximum file size?

Referencing data blocks via pointers

- If we need bigger files we can use an additional indirect pointer which is a pointer to a data block, filled with pointers
- A 4kB block with 4 byte pointers = 1024 pointers in a block
- If we have 12 direct pointers and 1 indirect pointer, what is the maximum file size?
    - $(12+1024) \star 4\text{kB} = 4144\text{ kB}$ (4 MB)

Multi-level indexing

- We can continue the process of using indirect pointers for double or even triple indirect pointers
- In a double indirect pointer, we reference a block that contains pointers to indirect blocks
- Those indirect blocks in turn contain the actual block addressed on disk
- With a double indirect pointer, we can achieve $1024^2 \star$ 4KB or 4GB files

Why have a set of direct pointers at all?

- Performing the extra steps of indirection to associate all the necessary block of data for a file isn't exactly efficient
- We are optimizing for the typical case
- If we can reference all the blocks we need with a small set of direct pointers, this is more efficient

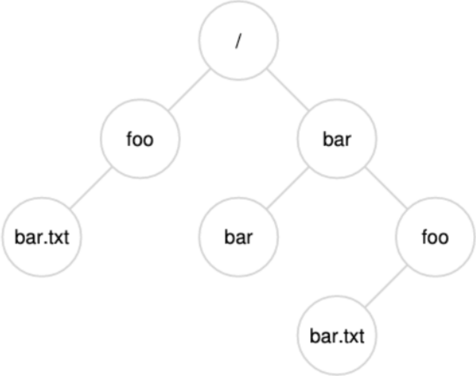
What about directories?

- Directories are often treated like special files
- They are just a set of data entries that are names + inode numbers
- Usually we find the inode number of a directory in its parent directory

inum	reclen	strlen	name
5	4	2	.
2	4	3	..
12	4	4	foo
13	4	4	bar
24	8	7	foobar

Directory hierarchy example

- / (0)
  - (foo, 1)
  - (bar, 2)
- foo (1)
  - (bar.txt, 3)
- bar (2)
  - (bar, 4)
  - (foo, 5)
- bar (4)
- foo (5)
  - (bar.txt, 6)



Question

How can you find the inode for the root directory?

A. Look in the superblock or in a well-known inode

B. Scan all the nodes

C. Scan all the data blocks

Putting it All Together

Let's walk through sample operations, and see how those operations translated into accesses to our reference file system's data structures

Example: Reading

- Let's try and read a file: /foo/bar
- Assume file is 12 kB (3 data blocks)
- We have to traverse directories and inodes
- We have to also write the last accessed time

File read timeline (1)

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

File read timeline (1)

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

Why must read for bar inode?

File read timeline (2)

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

File read timeline (2)

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

File read timeline (2)

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

Why must write for bar inode?

File read timeline (3)

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

File read timeline (3)

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

File read timeline (3)

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

File read timeline (3)

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

Example: Creating/Writing a File

- When creating we have to do lots of writes!
- We have to write to the inode allocation bitmap and to the directory, etc.
- We also have to allocate data blocks for the file we want to write and update the inode with that mapping as we go

File creation timeline (1)

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

File creation timeline (1)

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

Why must read for bar inode?

File creation timeline (2)

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

File creation timeline (2)

	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					