Lecture S4: File system – data layout, naming

****

Review -- 1 min

****

Intro to I/O -- overhead, latency, BW
Disks – avoid seeks, rotations
Header – given file header find rest of file
    Contiguous allocation
    Linked allocation
    FAT
    Indexed allocation
    Multi-level index

****

Outline - 1 min

****

Data layout
- files: file header, locating the rest of the file

Contiguous allocation
    Linked allocation
    FAT
    Indexed allocation
    Multi-level index

- naming, directories – given name, find header

****

Preview - 1 min

****

Wednesday: Guest lecture – IBM JFS and volume mgr
Next time: finish file systems
    In-memory data structures
    Scheduling
    Transactions

****

Lecture - 35 min

****

1. Naming and directories

ok: once I find a file header, I can find the rest of the file
How do you find a file header? **Name lookup**

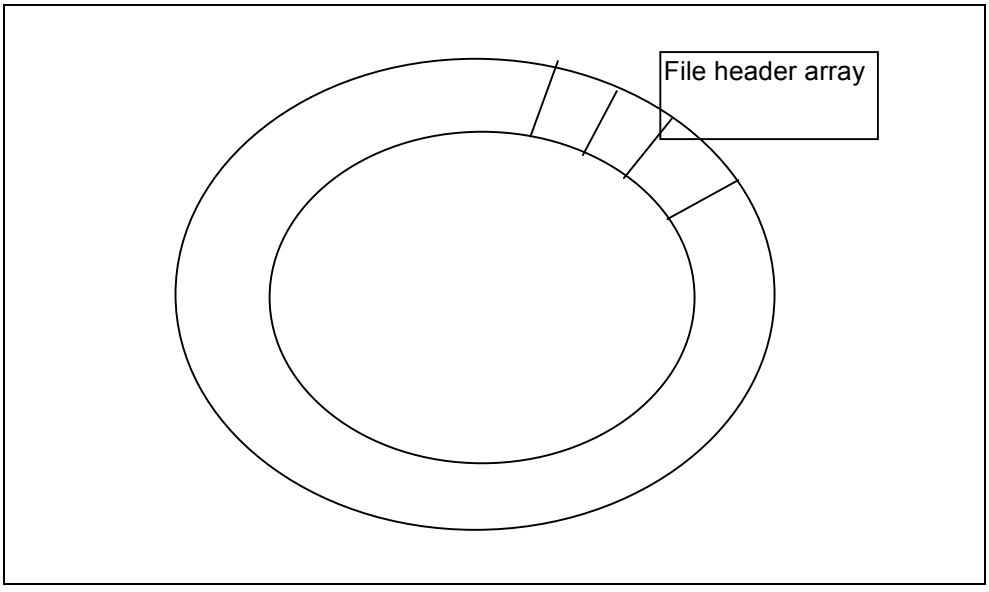
Directory: table of
<name> <pointer to file header>

1.1 File header

File header =
(1) root of data structure that allows you to get to all blocks of file (linked: pointer to first block; FAT: first entry; index: array of pointers; multi-level index: root node of tree; NTFS: array of pointers to extents)
(2) file metadata – owner, group, size, last accessed, last modified, permissions…

Where is file header stored on disk?

In (early) unix – special array in outermost cylinders:



Unix refers to file by index into array – tells it where to find the file header
UNIX-isms
"I-node" – file header
"I-number" – index into array
i-number is unique identifier for file

Unix file header organization seems strange
1) header not anywhere near data blocks. To read a small file, seek to get header, seek back to data
2) Fixed size, set when disk formatted. Means maximum # files that can be created

Why not put headers near data?
+ reliability – whatever happens to the disk, you can find all of the files
+ Unix BSD 4.2 puts portion of the file header array on each cylinder. For small directories – can fit all data, file headers, etc in same cylinder – no seeks!
+ file headers are much smaller than a whole block (a few hundred bytes), so multiple file headers fetched from disk at same time

QUESTION: do you ever look at a file header w/o reading the file? If not – put the file header in first block of the file!

Turns out – fetching the file header is about 4x more common in Unix than reading the file (ls, make, etc)

Bottom line:
array of headers (array of inodes)
index of array ("inumber") → header → bytes of a file

1.2 Naming

1.2.1 Options

1. Use index (ask user to specify I-node number)
2. text name
3. icon

With icon or text, need to map name→index

1.2.2 Directories

Directory maps name→ file index (where to find file header)

Directory just a table of file name, file index pairs
I could write it on a piece of paper and carry it around in my pocket…

Directory is just a file
Only OS permitted to modify directory
- ♦ so it always contains name→file index

Any program can read directory file
this is how ls works

1.2.3 Directory hierarchy

Take this one step at a time, starting at the bottom:

Ok – how do you find blocks of a file?
→ find its header – header has pointers to blocks of a file
how do you find a header?
→ find its I-number – inumber is pointer to header
how do you find a file's inumber?
→ read its directory – directory maps name to inumber
But wait, directory is a file – how do you find *it*?

How do you find a file?
→ find its header
…

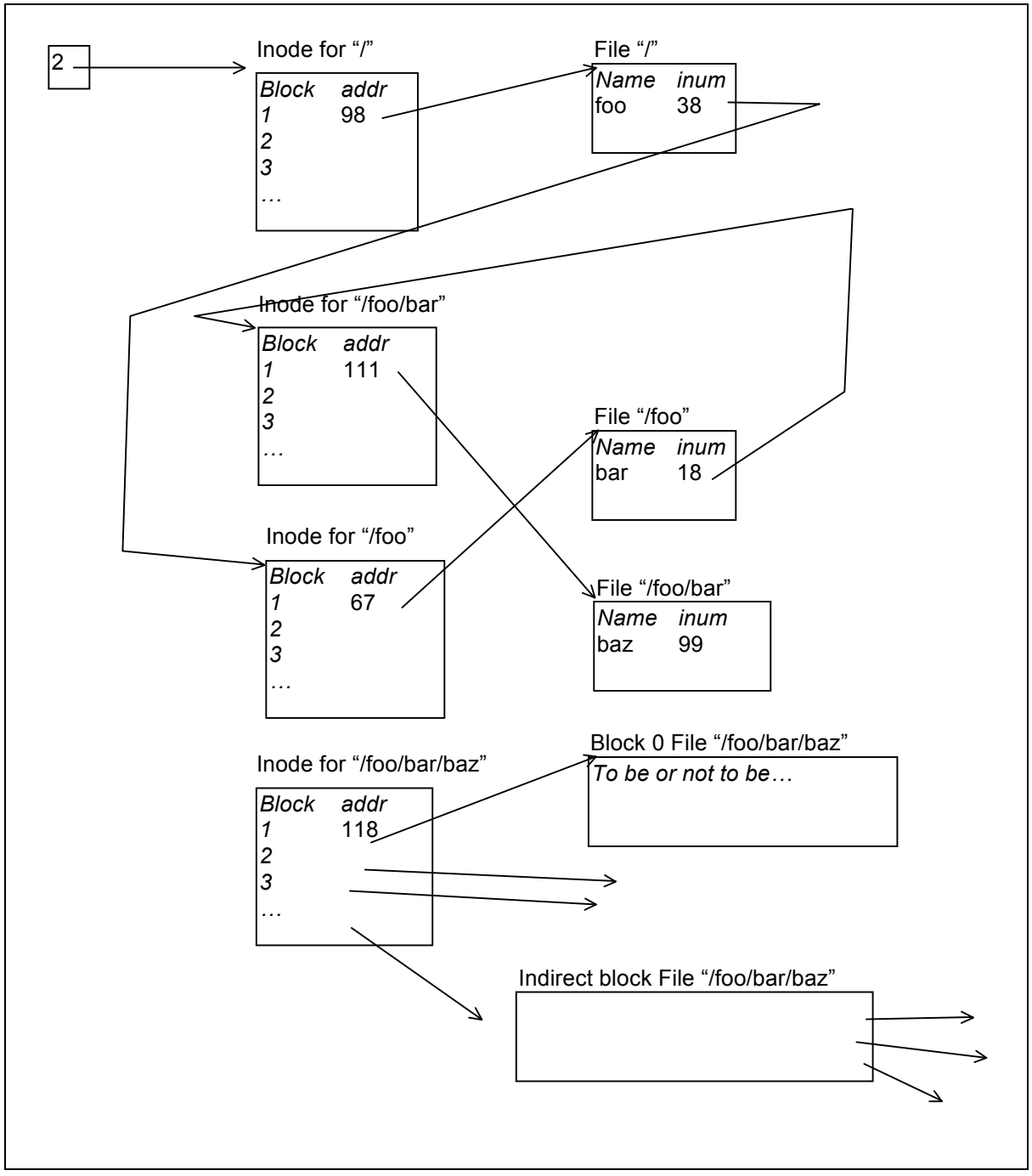
You're seniors – a little recursion shouldn't bother you!
→ lets you nest directories – directories of directories, etc

Interpret:
/foo/bar/baz

- baz is a file
- bar/ is a directory that contains the inumber of file baz
- foo/ is a directory that contains the inumber of file bar

What do you need with recursion? A base case

- "/" is a directory that contains the inumber of file foo
- the inumber of "/" is 2



How many disk I/Os needed to access first byte of file
1) read in file header for root (always from fixed location)
2) read in first data block for root
3) read in file header for foo
4) read in first block of foo
5) read in file header for bar
6) read in first block of bar
7) read in file header for baz
8) read in first data block for baz

How can this possibly be efficient?

(1) Caching

(2) current working directory: short cut for both user and system.
Each address space stores file index for current directory.
Allows user to specify relative file name instead of absolute path (if no leading "/")

Thus, to read first block of file, just last 4 steps above

(3) Open file table
Open a file using path name.
OS stores file number in a table, returns key to table entry ("file descriptor"
read/write calls use this key to identify the file

```
fd = open("foo"); // Open file foo in current working directory
read(fd, 512, buffer); // read 512 bytes of file into buffer
```

This, to read second block of file, just last 2 steps above

1.3 Links

Hard link

- store reference to same header in two different directories
- QUESTION: what happens on delete?
  - [→keep reference count in header]

Soft link
-- store path name in directory entry

QUESTION: Why is it hard to implement hard links in FAT?

****

Summary - 1 min

****