Lecture 18: Naming and Directories

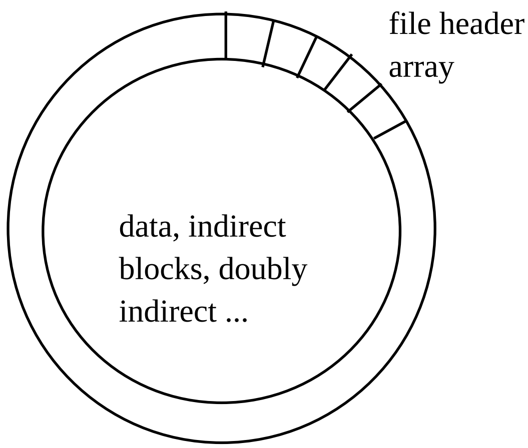
18.1 Main Points

How do users name files? What is a name?

Lookup: given a name, how do you translate it into a file header?

18.2 File Header Storage

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 the array

UNIX file header organization, seems strange:

1. header not anywhere near the data blocks. To read a small file, seek to get header, seek back to data.

2. Fixed size, set when disk is formatted. Means maximum number of 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 without reading the file?

18.3 Naming

18.3.1 Options

1. use index (ask users specify i-node number). Easier for system, not as easy for users.

2. text name

3. icon

With icons or text, still have to map name -> index

18.3.2 Directories

Directory maps name -> file index (where to find file header)

Directory just a table of file name, file index pairs.

General idea: **relation**. Table associating things together.

Principle behind relational databases. Relations are useful because of associative match -- look up based on content

For example: (employee name, salary, address, supervisor).
(flight, passenger, seat assignment, agent)
Directories just a special kind of a relation, connecting file name to index (ditto with password file, caches, etc.)

Each directory is stored as a file, containing a list of "name", index pairs.

But, only OS permitted to modify directory

Any program can read the directory file. This is how "ls" works.

Problem: means hard to change file directory structure!

18.3.3 Directory Hierarchy

Directories organized into hierarchical structure

```
/joe/abcde/file1
^ root
  ^ subdir joe
       ^subdir abcde
```

Top-level directory has pair: <joe, #>. joe has pair <abcde, #>, etc.

How many disk I/O's to access first byte of file1?

1. Read in file header for root (always at fixed spot on disk).

2. Read in first data block for root.

3. Read in file header for joe

4. Read in first data block for joe.

5. Read in file header for abcde

6. Read in first data block for abcde.

7. Read in file header for file1

8. Read in first data block for file1

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

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

How can this possibly be efficient? Caching!