CS162
Operating Systems and Systems Programming
Lecture 18

File Systems

April 3rd, 2017

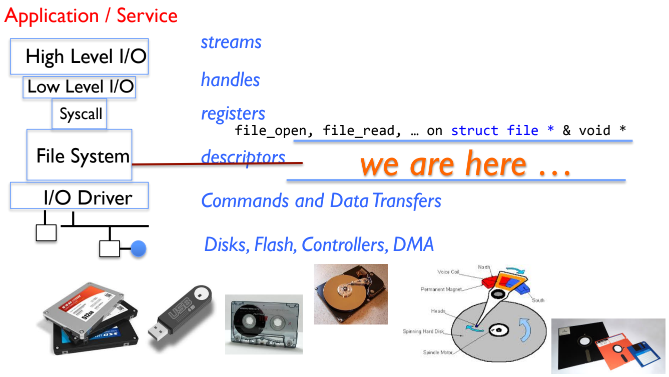
Prof. Ion Stoica

http://cs162.eecs.Berkeley.edu

Recall: How do we Hide I/O Latency?

- Blocking Interface: "Wait"
  - When request data (e.g., read() system call), put process to sleep until data is ready
  - When write data (e.g., write() system call), put process to sleep until device is ready for data
- Non-blocking Interface: "Don't Wait"
  - Returns quickly from read or write request with count of bytes successfully transferred to kernel
  - Read may return nothing, write may write nothing
- Asynchronous Interface: "Tell Me Later"
  - When requesting data, take pointer to user's buffer, return immediately; later kernel fills buffer and notifies user
  - When sending data, take pointer to user's buffer, return immediately; later kernel takes data and notifies user

I/O & Storage Layers



Operations, Entities and Interface

Application / Service

High Level I/O — *streams*

Low Level I/O — *handles*

Syscall — *registers*

`file_open, file_read, … on struct file * & void *`

File System — *descriptors*

we are here …

I/O Driver — *Commands and Data Transfers*

Disks, Flash, Controllers, DMA

Recall: C Low level I/O

- Operations on File Descriptors – as OS object representing the state of a file
  - User has a "handle" on the descriptor

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int open (const char *filename, int flags [, mode_t mode])
int create (const char *filename, mode_t mode)
int close (int filedes)
```

Bit vector of:
- Access modes (Rd,Wr, …)
- Open Flags (Create, …)
- Operating modes (Appends, …)

Bit vector of Permission Bits:
- User|Group|Other X R|W|X

http://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html

Recall: C Low Level Operations

```
ssize_t read (int filedes, void *buffer, size_t maxsize)
 - returns bytes read, 0 => EOF, -1 => error
ssize_t write (int filedes, const void *buffer, size_t size)
 - returns bytes written
off_t lseek (int filedes, off_t offset, int whence)
 - set the file offset
   * if whence == SEEK_SET: set file offset to “offset”
   * if whence == SEEK_CRT: set file offset to crt location + “offset”
   * if whence == SEEK_END: set file offset to file size + “offset”
int fsync (int fildes)
 - wait for i/o of filedes to finish and commit to disk
void sync (void) - wait for ALL to finish and commit to disk
```

- When write returns, data is on its way to disk and can be read, but it may not actually be permanent!

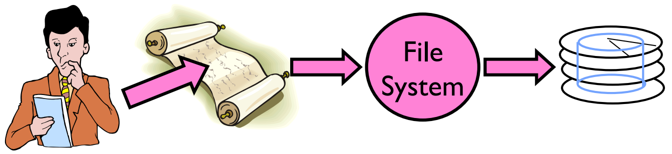
Building a File System

- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
- File System Components
  - **Naming**: Interface to find files by name, not by blocks
  - **Disk Management**: collecting disk blocks into files
  - **Protection**: Layers to keep data secure
  - **Reliability/Durability**: Keeping of files durable despite crashes, media failures, attacks, etc.

Recall: User vs. System View of a File

- User's view:
  - Durable Data Structures
- System's view (system call interface):
  - Collection of Bytes (UNIX)
  - Doesn't matter to system what kind of data structures you want to store on disk!
- System's view (inside OS):
  - Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
  - Block size ≥ sector size; in UNIX, block size is 4KB

Recall: Translating from User to System View



- What happens if user says: give me bytes 2—12?
  - Fetch block corresponding to those bytes
  - Return just the correct portion of the block
- What about: write bytes 2—12?
  - Fetch block
  - Modify portion
  - Write out Block
- Everything inside File System is in whole size blocks
  - For example, `getc()`, `putc()` ⇒ buffers something like 4096 bytes, even if interface is one byte at a time
- From now on, file is a collection of blocks

Disk Management Policies (1/2)

- Basic entities on a disk:
  - File: user-visible group of blocks arranged sequentially in logical space
  - Directory: user-visible index mapping names to files
- Access disk as linear array of sectors. Two Options:
  - Identify sectors as vectors [cylinder, surface, sector], sort in cylinder-major order, not used anymore
  - Logical Block Addressing (LBA): Every sector has integer address from zero up to max number of sectors
  - Controller translates from address ⇒ physical position
    - » First case: OS/BIOS must deal with bad sectors
    - » Second case: hardware shields OS from structure of disk

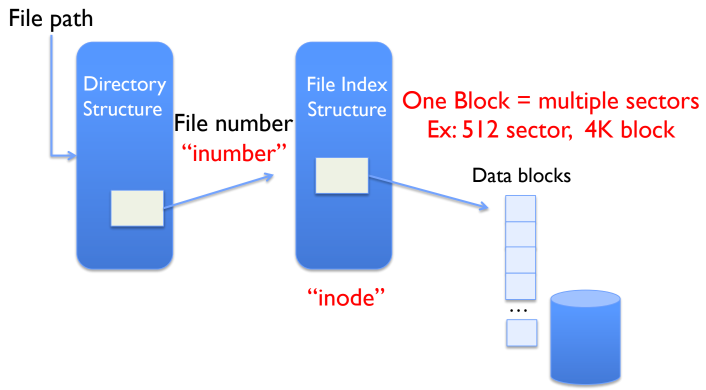
Recall: Disk Management Policies (2/2)

- Need way to track free disk blocks
  - Link free blocks together ⇒ too slow today
  - Use bitmap to represent free space on disk
- Need way to structure files: File Header
  - Track which blocks belong at which offsets within the logical file structure
  - Optimize placement of files' disk blocks to match access and usage patterns

Designing a File System …

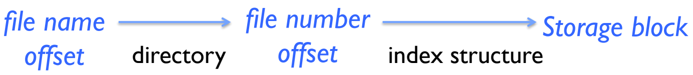
- What factors are critical to the design choices?
- Durable data store => it's all on disk
- (Hard) Disks Performance !!!
  - Maximize sequential access, minimize seeks
- Open before Read/Write
  - Can perform protection checks and look up where the actual file resource are, in advance
- Size is determined as they are used !!!
  - Can write (or read zeros) to expand the file
  - Start small and grow, need to make room
- Organized into directories
  - What data structure (on disk) for that?
- Need to allocate / free blocks
  - Such that access remains efficient

Components of a File System



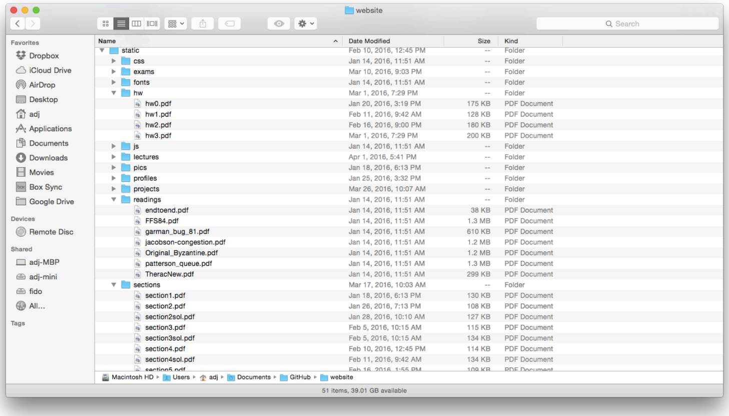
One Block = multiple sectors
Ex: 512 sector, 4K block

Components of a file system



- Open performs *Name Resolution*
  - Translates pathname into a "file number"
    - Used as an "index" to locate the blocks
  - Creates a file descriptor in PCB within kernel
  - Returns a "handle" (another integer) to user process
- Read, Write, Seek, and Sync operate on handle
  - Mapped to file descriptor and to blocks

Directories



Name	Date Modified	Size	Kind
static	Feb 10, 2016, 12:45 PM	--	Folder
css	Jan 14, 2016, 11:51 AM	--	Folder
exams	Mar 10, 2016, 9:03 PM	--	Folder
fonts	Jan 14, 2016, 11:51 AM	--	Folder
hw	Mar 1, 2016, 7:29 PM	--	Folder
hw0.pdf	Jan 20, 2016, 3:19 PM	175 KB	PDF Document
hw1.pdf	Feb 11, 2016, 9:42 AM	128 KB	PDF Document
hw2.pdf	Feb 16, 2016, 9:00 PM	180 KB	PDF Document
hw3.pdf	Mar 1, 2016, 7:29 PM	200 KB	PDF Document
js	Jan 14, 2016, 11:51 AM	--	Folder
lectures	Apr 1, 2016, 5:41 PM	--	Folder
pics	Jan 18, 2016, 6:13 PM	--	Folder
profiles	Jan 25, 2016, 3:32 PM	--	Folder
projects	Mar 26, 2016, 10:07 AM	--	Folder
readings	Jan 14, 2016, 11:51 AM	--	Folder
endtoend.pdf	Jan 14, 2016, 11:51 AM	38 KB	PDF Document
FFS84.pdf	Jan 14, 2016, 11:51 AM	1.3 MB	PDF Document
garman_bug_81.pdf	Jan 14, 2016, 11:51 AM	610 KB	PDF Document
jacobson-congestion.pdf	Jan 14, 2016, 11:51 AM	1.2 MB	PDF Document
Original_Byzantine.pdf	Jan 14, 2016, 11:51 AM	1.2 MB	PDF Document
patterson_queue.pdf	Jan 14, 2016, 11:51 AM	1.3 MB	PDF Document
TheracNew.pdf	Jan 14, 2016, 11:51 AM	299 KB	PDF Document
sections	Mar 17, 2016, 10:03 AM	--	Folder
section1.pdf	Jan 18, 2016, 6:13 PM	130 KB	PDF Document
section2.pdf	Jan 26, 2016, 7:13 PM	108 KB	PDF Document
section2sol.pdf	Jan 28, 2016, 10:10 AM	127 KB	PDF Document
section3.pdf	Feb 5, 2016, 10:15 AM	115 KB	PDF Document
section3sol.pdf	Feb 5, 2016, 10:15 AM	134 KB	PDF Document
section4.pdf	Feb 10, 2016, 12:45 PM	114 KB	PDF Document
section4sol.pdf	Feb 11, 2016, 9:42 AM	134 KB	PDF Document
section5.pdf	Feb 16, 2016, 1:55 PM	109 KB	PDF Document

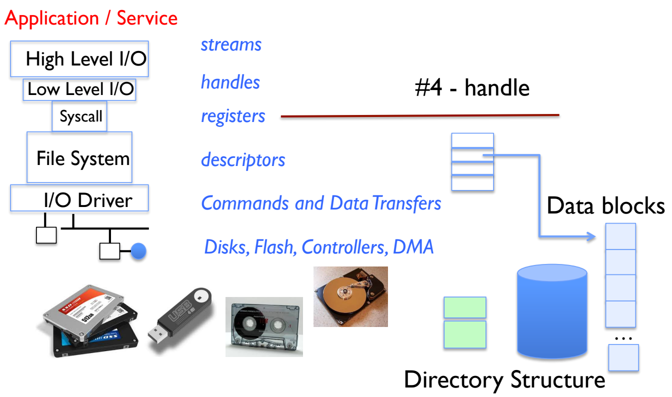
Macintosh HD ▸ Users ▸ adj ▸ Documents ▸ GitHub ▸ website

51 items, 39.01 GB available

Directory

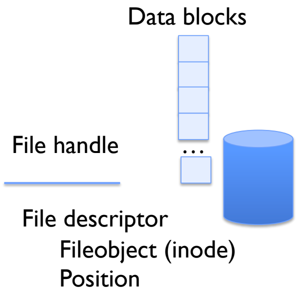
- Basically a hierarchical structure
- Each directory entry is a collection of
  - Files
  - Directories
    - A link to another entries
- Each has a name and attributes
  - Files have data
- Links (hard links) make it a DAG, not just a tree
  - Softlinks (aliases) are another name for an entry

I/O & Storage Layers

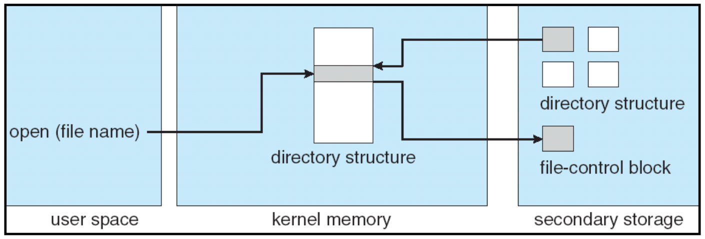


File

- Named permanent storage
- Contains
  - Data
    - » Blocks on disk somewhere
  - Metadata (Attributes)
    - » Owner, size, last opened, …
    - » Access rights
      - •R, W, X
      - •Owner, Group, Other (in Unix systems)
      - •Access control list in Windows system

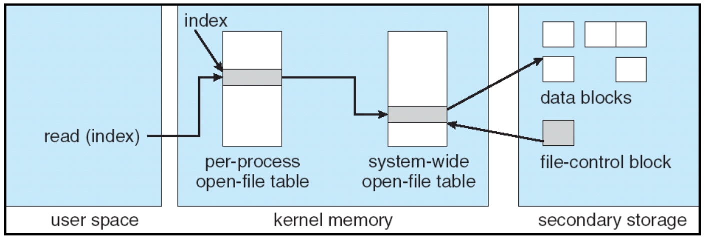


In-Memory File System Structures



- Open system call:
  - Resolves file name, finds file control block (inode)
  - Makes entries in per-process and system-wide tables
  - Returns index (called "file handle") in open-file table

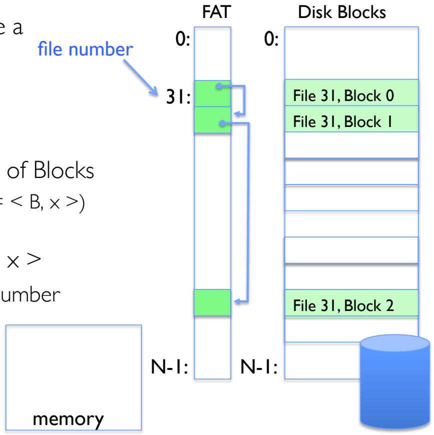
In-Memory File System Structures

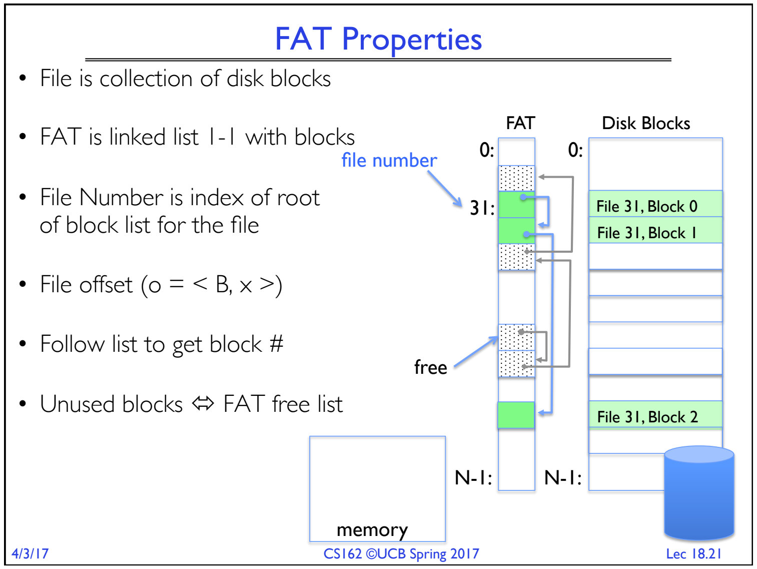


- Read/write system calls:
  - Use file handle to locate inode
  - Perform appropriate reads or writes

Our first filesystem: FAT (File Allocation Table)

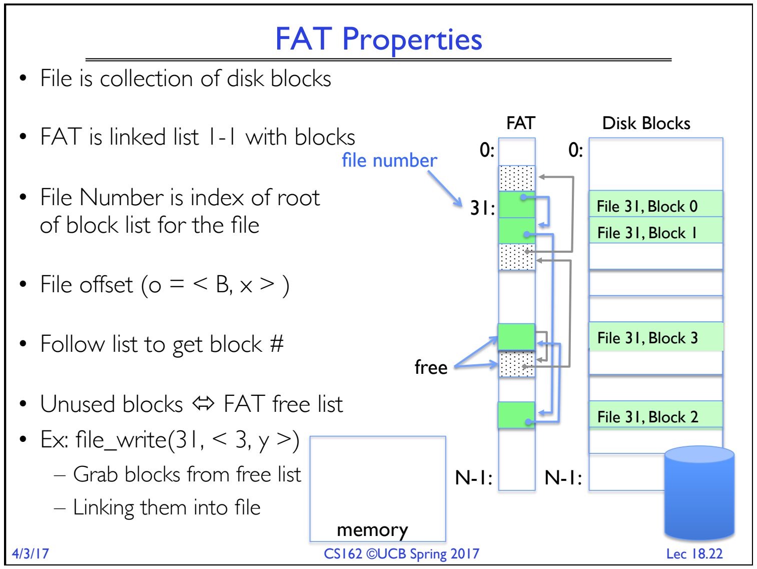
- The most commonly used filesystem in the world!
- Assume (for now) we have a way to translate a path to a "file number"
  - i.e., a directory structure
- Disk Storage is a collection of Blocks
  - Just hold file data (offset o = < B, x >)
- Example: file_read 31, < 2, x >
  - Index into FAT with file number
  - Follow linked list to block
  - Read the block from disk into memory





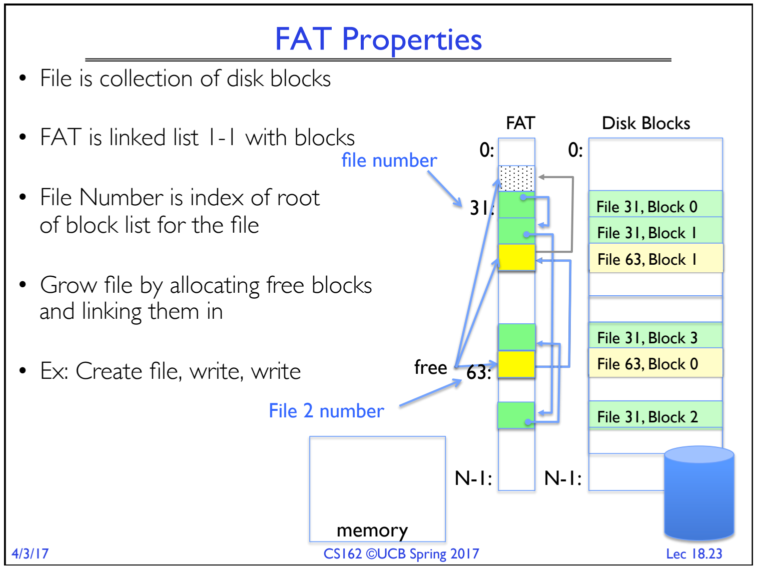
FAT Properties

- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- File offset (o = < B, x >)
- Follow list to get block #
- Unused blocks ⇔ FAT free list



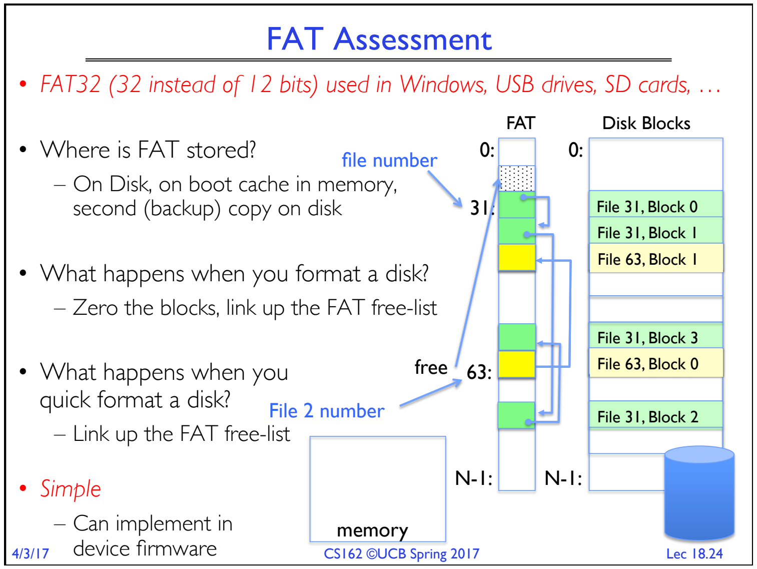
FAT Properties

- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- File offset (o = < B, x >)
- Follow list to get block #
- Unused blocks ⇔ FAT free list
- Ex: file_write(31, < 3, y >)
  - Grab blocks from free list
  - Linking them into file



FAT Properties

- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- Grow file by allocating free blocks and linking them in
- Ex: Create file, write, write

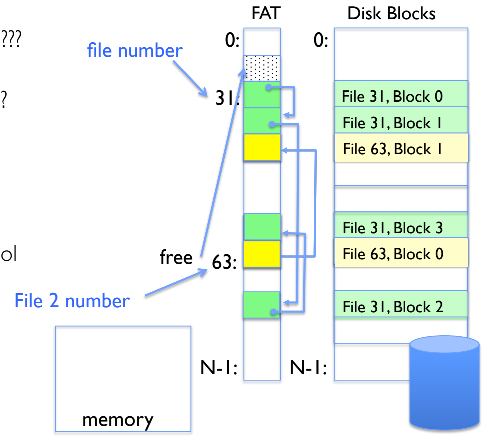


FAT Assessment

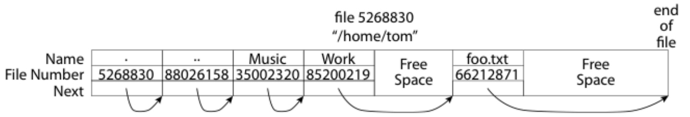
- *FAT32 (32 instead of 12 bits) used in Windows, USB drives, SD cards, …*
- Where is FAT stored?
  - On Disk, on boot cache in memory, second (backup) copy on disk
- What happens when you format a disk?
  - Zero the blocks, link up the FAT free-list
- What happens when you quick format a disk?
  - Link up the FAT free-list
- *Simple*
  - Can implement in device firmware

FAT Assessment – Issues

- Time to find block (large files) ??
- Block layout for file ???
- Sequential Access ???
- Random Access ???
- Fragmentation ???
  - MSDOS defrag tool
- Small files ???
- Big files ???



What about the Directory?



- Essentially a file containing
  <file_name: file_number> mappings
- Free space for new entries
- In FAT: file attributes are kept in directory (!!!)
- Each directory a linked list of entries
- Where do you find root directory ("/")?

Directory Structure (cont'd)

- How many disk accesses to resolve "/my/book/count"?
  - Read in file header for root (fixed spot on disk)
  - Read in first data block for root
    - » Table of file name/index pairs. Search linearly – ok since directories typically very small
  - Read in file header for "my"
  - Read in first data block for "my"; search for "book"
  - Read in file header for "book"
  - Read in first data block for "book"; search for "count"
  - Read in file header for "count"
- Current working directory: Per-address-space pointer to a directory (inode) used for resolving file names
  - Allows user to specify relative filename instead of absolute path (say CWD="/my/book" can resolve "count")

Many Huge FAT Security Holes!

- FAT has no access rights
- FAT has no header in the file blocks
- Just gives an index into the FAT
  - (file number = block number)

Administrivia

- Project 2
  - Code due today Monday 4/3
  - Final report due Wednesday 4/5

BREAK

Characteristics of Files

A Five-Year Study of File-System Metadata

NITIN AGRAWAL
University of Wisconsin, Madison
and
WILLIAM J. BOLOSKY, JOHN R. DOUCEUR, and JACOB R. LORCH
Microsoft Research

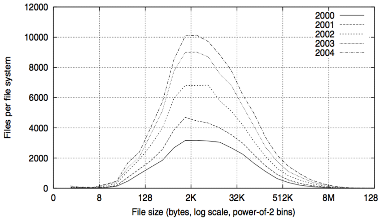


Fig. 2. Histograms of files by size.

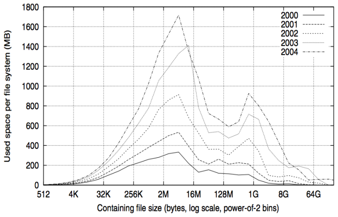


Fig. 4. Histograms of bytes by containing file size.

Characteristics of Files

- Most files are small, growing numbers of files over

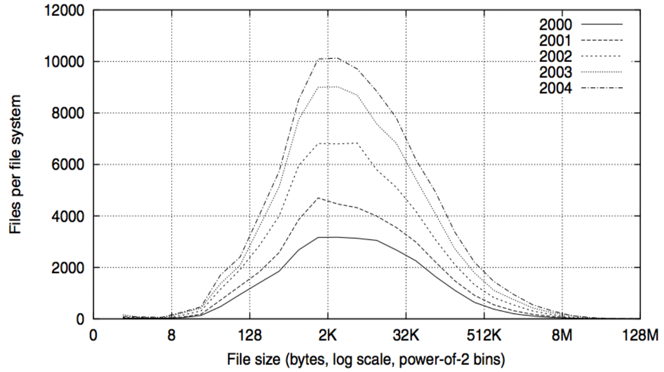


Fig. 2. Histograms of files by size.

Characteristics of Files

- Most of the space is occupied by the rare big ones

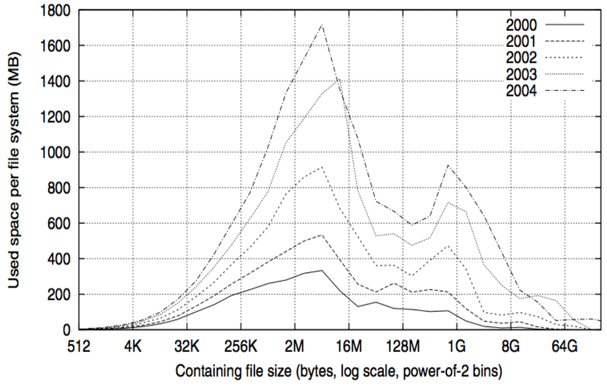
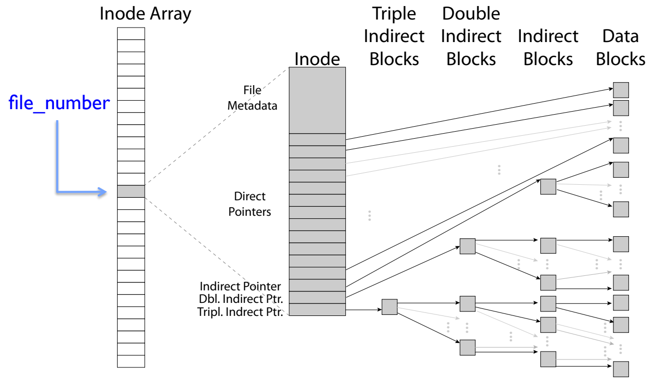


Fig. 4. Histograms of bytes by containing file size.

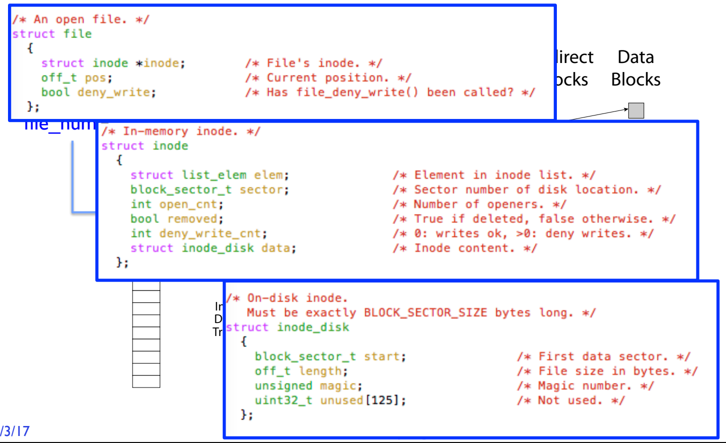
So What About a "Real" File System?

- Meet the inode:



An "Almost Real" File System

- Pintos: `src/filesys/file.c`, `inode.c`



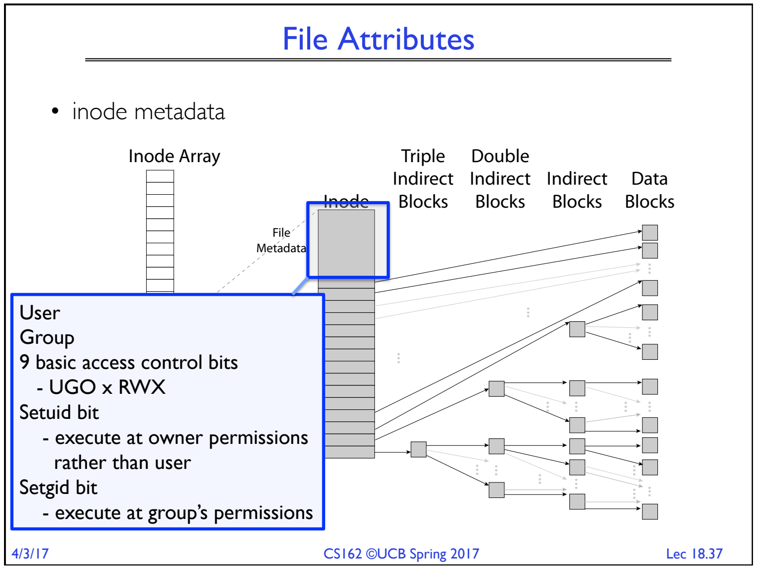
```
/* An open file. */
struct file
  {
    struct inode *inode;        /* File's inode. */
    off_t pos;                  /* Current position. */
    bool deny_write;            /* Has file_deny_write() been called? */
  };
```

```
/* In-memory inode. */
struct inode
  {
    struct list_elem elem;              /* Element in inode list. */
    block_sector_t sector;              /* Sector number of disk location. */
    int open_cnt;                       /* Number of openers. */
    bool removed;                       /* True if deleted, false otherwise. */
    int deny_write_cnt;                 /* 0: writes ok, >0: deny writes. */
    struct inode_disk data;             /* Inode content. */
  };
```

```
/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start;               /* First data sector. */
    off_t length;                       /* File size in bytes. */
    unsigned magic;                     /* Magic number. */
    uint32_t unused[125];               /* Not used. */
  };
```

Unix File System

- Original inode format appeared in BSD 4.1
  - Berkeley Standard Distribution Unix
  - Part of your heritage!
  - Similar structure for Linux Ext2/3
- File Number is index into inode arrays
- Multi-level index structure
  - Great for little and large files
  - Asymmetric tree with fixed sized blocks
- Metadata associated with the file
  - Rather than in the directory that points to it
- UNIX Fast File System (FFS) BSD 4.2 Locality Heuristics:
  - Block group placement
  - Reserve space
- Scalable directory structure



File Attributes

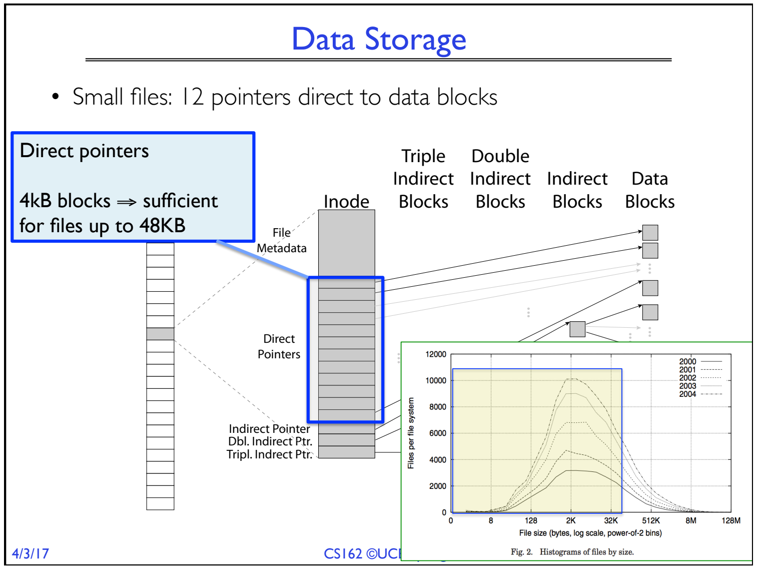
- inode metadata

User
Group
9 basic access control bits
- UGO x RWX

Setuid bit
- execute at owner permissions rather than user

Setgid bit
- execute at group's permissions

4/3/17 CS162 ©UCB Spring 2017 Lec 18.37



Data Storage

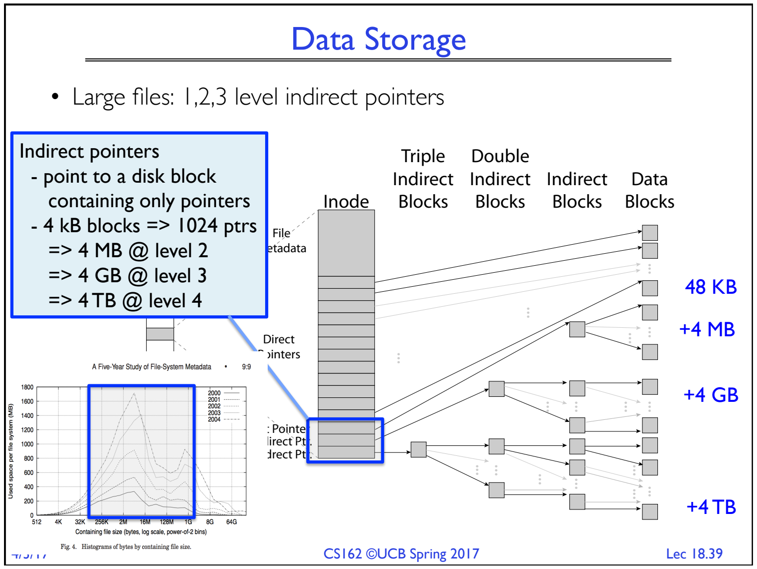
- Small files: 12 pointers direct to data blocks

Direct pointers

4kB blocks ⇒ sufficient for files up to 48KB

Fig. 2. Histograms of files by size.

4/3/17 CS162 ©UCB Spring 2017



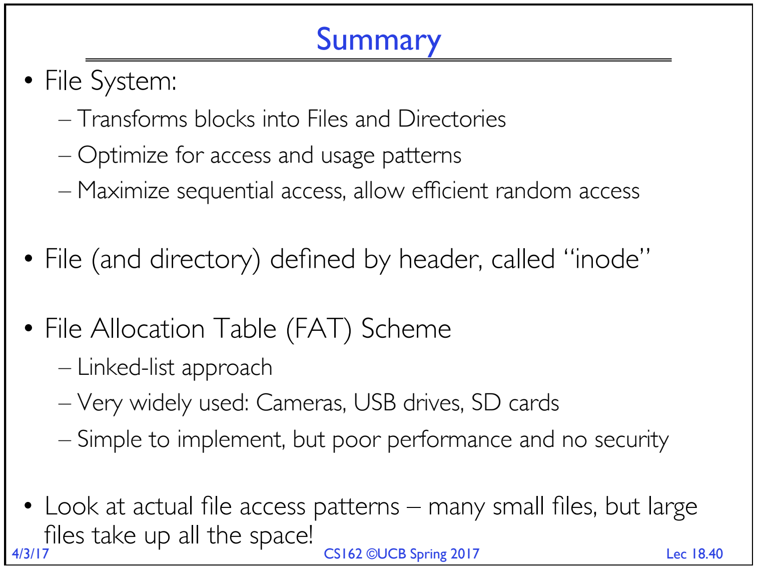
Data Storage

- Large files: 1,2,3 level indirect pointers

Indirect pointers
- point to a disk block containing only pointers
- 4 kB blocks => 1024 ptrs
  - => 4 MB @ level 2
  - => 4 GB @ level 3
  - => 4 TB @ level 4

Fig. 4. Histograms of bytes by containing file size.

4/3/17 CS162 ©UCB Spring 2017 Lec 18.39



Summary

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for access and usage patterns
  - Maximize sequential access, allow efficient random access
- File (and directory) defined by header, called "inode"
- File Allocation Table (FAT) Scheme
  - Linked-list approach
  - Very widely used: Cameras, USB drives, SD cards
  - Simple to implement, but poor performance and no security
- Look at actual file access patterns – many small files, but large files take up all the space!

4/3/17 CS162 ©UCB Spring 2017 Lec 18.40