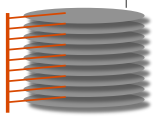
Introduction to File Systems: FS Structure, Disk Layout

ECE 595
Nov 1

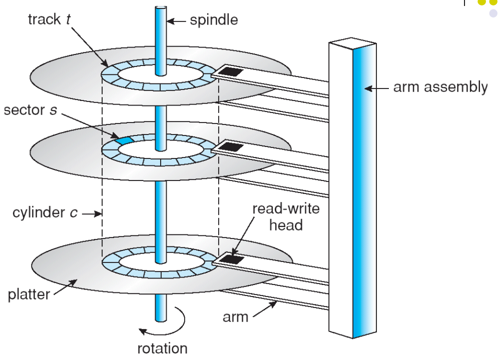
Y. Charlie Hu

Disk



- Platter / Head / Tracks / Sectors / Cylinders
- 256 heads * 17849 tracks * 63 sectors * 512 bytes = 140 GB disk

Moving-head Disk Mechanism



Overview of Disk (non-volatile storage)

- Magnetic disks provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 200 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash**: disk head making contact w/ disk surface, not repairable
- Disks can be removable (hot swap)
- Drive attached to computer via **I/O bus**
  - Buses vary, including **EIDE, ATA, SATA, USB, SCSI**
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

Disk formatting

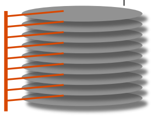
- *Low-level formatting*, or *physical formatting* — Dividing a disk into sectors (logical blocks) that the disk controller can read and write
  - 512 bytes, 1024 bytes
- Handling bad blocks (extra-level of indirection!)
  - IDE drives: "manually" scan the disk and mask out bad blocks in file system management
  - SCSI drives: disk controller maintains and masks off bad blocks
    - *Sector sparing*
      - Low-level formatting sets aside spare sectors
      - Controller can replace bad blocks logically with a spare one

Disk Accesses

- Modern disk drives are addressed as large 1D arrays of logical blocks, where logical blocks are smallest transfer units
  - The size of logical blocks is usually 512 bytes
  - The logical blocks have high reliability (e.g. SCSI)
  - File system often has its own unit, called file system block, that is a multiple of disk blocks
    - e.g. 4K bytes

Disk

"My lab3 solution is in ~ece595/lab3/process.c."



"I will save my lab4 solution on platter 5, track 8739, sector 3, offset 10 bytes."

- Platter / Head / Tracks / Sectors / Cylinders
- 256 heads * 17849 tracks * 63 sectors * 512 bytes = 140 GB disk

File System

- The most visible aspect of an operating system
  - Files
  - Directories
  - Protections
  - Persistence
  - Transparent remote access

What is a File?

- File: a named collection of bytes stored on disk
  - From OS’s standpoint
    - A file consists of a bunch of blocks stored on disk
  - From programmer’s view
    - A collection of data records
  - File system performs the magic / translation
    - Pack bytes into disk blocks on writing
    - Unpack them again on reading

[lec1] Why Files?

- Physical reality
  - Block oriented
  - Physical sector numbers
  - No protection among users of the system
  - Data might be corrupted if machine crashes
- File system abstraction
  - Byte oriented
  - Named files
  - Users protected from each other
  - Robust to machine failures

File Types

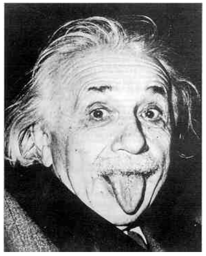
- ASCII – plain text
  - inbox
- A Unix executable file (a.out)
  - header: magic number, sizes, entry point, flags
  - text (code)
  - data
- Devices
  - keyboard
  - terminal

Common Addressing Patterns

- *Sequential*: information is processed in order, one piece after another
  - Example?
- *Random (direct)*: can access any block in the file directly without passing through its predecessors
  - E.g. databases
- *Content based*: search for blocks with particular values
  - E.g. hash table, dictionary

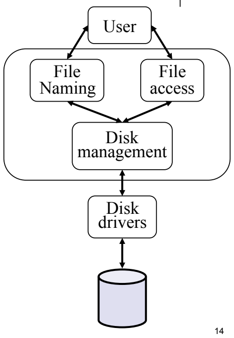
So What Makes File Systems Hard?

- Files grow and shrink
  - Little *a priori* knowledge
  - 6~8 orders of magnitude in file sizes
- Overcoming disk performance behavior
  - Highly nonuniform access
  - Desire for efficiency
- Coping with failure



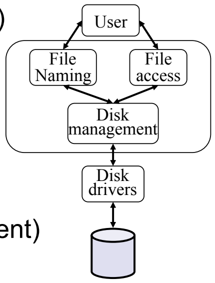
File System Components

- Naming/Access
  - User gives file name, not track or sector number, to locate data
- Disk management
  - Arrange collection of disk blocks into files
- Security
  - Keep information secure
- Reliability/durability
  - When system crashes, lose stuff in memory, but want files to be durable



Roadmap (The Journey)

- Functionality (naming/access)
  - Data structures
  - Files/directories
  - File operations
- Performance (disk management)
  - Disk layout (I/O performance)
  - Disk scheduling (I/O performance)



Definitions

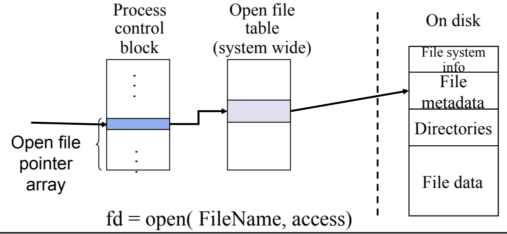
- *File descriptor* (fd) – an integer used to represent a file – easier than using names
- *Metadata* – bookkeeping data that describes the file or info about it; not the actual content
  - *inode* – "index node", file metadata on Unix
  - inode design discussed in next lecture
- *Open file table* – in-memory, system-wide list of file metadata in use

File System API

- OS provides the file system abstraction
- How do application processes access the file system?

Opening a File: fd = open("file1")

- File name lookup and authentication
- Create an entry in the open file table (system wide) if it is not in
- Copy the file metadata to in-memory data structure, if it is not in
- Create an entry in PCB
- Link up the data structures
- Return a pointer to user

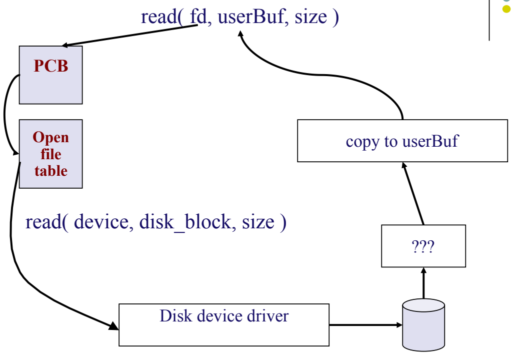


fd = open(FileName, access)

From User to System View

- What should FS do if user wants to read 10 bytes from a file starting at byte 2?
- What should FS do if user wants to write 10 bytes to a file starting at byte 2?

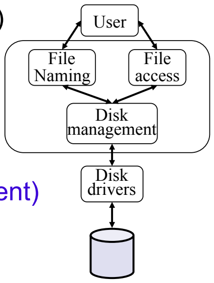
Reading a file (system call)



Modern disk drives are addressed as large one-dimensional arrays of logical blocks

Roadmap

- Functionality (naming/access)
  - Data structures
  - Files/directories
  - File operations
- Performance (disk management)
  - Disk layout (I/O performance)

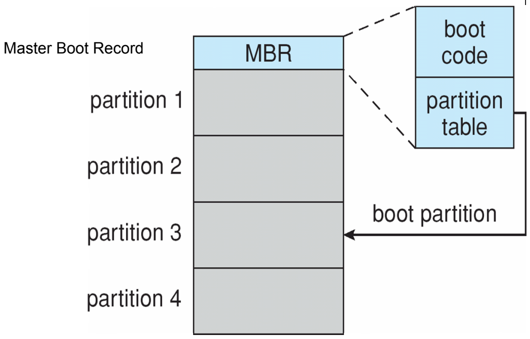


Boot block

extra level of indirection!

- The bootstrap program is stored in ROM
  - Chip needs no initialization
  - At fixed location
  - RO means never affected by virus
  - But bootstrap program cannot be changed
    - location of OS not flexible
- Instead, modern systems store a tiny bootloader in ROM
  - which loads a full bootstrap program stored at the boot block at a fixed location on disk
- The full bootstrap loads the OS from non-fixed location

Booting from a Disk in Windows 2000



A Disk Layout for A File System

Boot block	Super block	File metadata (i-node in Unix)	File data blocks

- Boot block: contains info to boot OS
- Superblock defines a file system
  - Size of the file system
  - Free metadata (inode) count and pointers
  - Free block count and pointers (or pointer to bitmap)
  - Location of the metadata of the root directory
- What if the superblock is corrupted?
  - What can we do?

Disk management: Data Structures (to keep track of)

- Used space on disk:
  - A “header” for each file (part of the file meta-data)
    - Point to Disk blocks associated with each file
- Free space on disk
  - Bitmap
    - 1 bit per block (sector)
    - blocks numbered in cylinder-major order (why?)
  - Linked list of free blocks
  - How much space does a bitmap need for a 4GB disk?
    - 4294967296 bytes → 8388608 sectors → 1MB bitmap

Lab4 – Part 1 – FS initialization and shutdown

- We provide you the raw disk I/O
  - Simulated using the real UNIX file system
  - DiskCreate()
  - DiskWriteBlock()
  - DiskReadBlock()
- You are asked to implement
  - NewFileSys()

Boot block	Super block	File metadata (i-node in Unix)	File data blocks

Lab4 – Part 2 – low-level FS operations

- OpenFileSys
- CloseFileSys()
- AllocateBlock(): allocate a file system block
- FreeBlock(): free a file system block
- ReadBlock(): read a file system block
- WriteBlock(): write a file system block
- …

Lab3 – Part 3 – inode based functions

- Dfs_inodeOpen()
- Dfs_inodeDelete()

- Dfs_inodeReadBytes()
- Dfs_inodeWriteBytes()

- Dfs_inodeAllocateVirtualBlock(handle, virtual_blocknum)
- Dfs_inodeTranslateVirutaltoFilesus(handle, birtual_blocknum)

Lab4 – Part 4 – basic file operations

- file_open()
- file_read()
- file_write()
- file_close()
- file_seek()
- file_chmod()
- file_delete()
- …

How to deal with concurrent accesses?

Lab5 – Part 5 – Multi-level directory or Buffer Cache

Reading

- Chapters 10-11