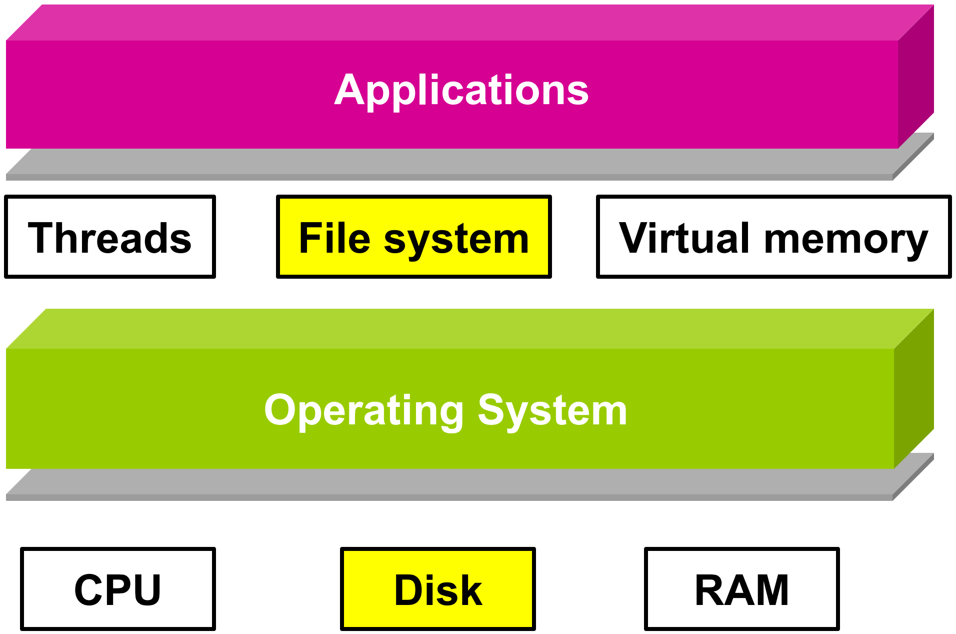
EECS 482
Introduction to Operating Systems

Winter 2018

Baris Kasikci

Slides by: Harsha V. Madhyastha

OS Abstractions



- Next few lectures:
  - What interface does file system export to apps?
  - How does file system interact with hardware?

Reality vs. Abstraction

Hardware interface	OS Abstraction
Heterogenous	Uniform
One/few storage objects (disk)	Many storage objects (files)
Simple naming (Numeric, Flat, Separate)	Rich naming (Symbolic, Structured, Unified)
Fixed block assignment	Flexible block assignment
Slow	Fast
Possible inconsistency on system crash	Crash consistency

Dealing with heterogeneity

- Problem: Wide range of disk types and interfaces
  - How to manage this diversity?
- Solution: Add device-driver abstraction inside OS

rest of OS and application programs

virtual machine interface

Yet another example of the power of abstractions!

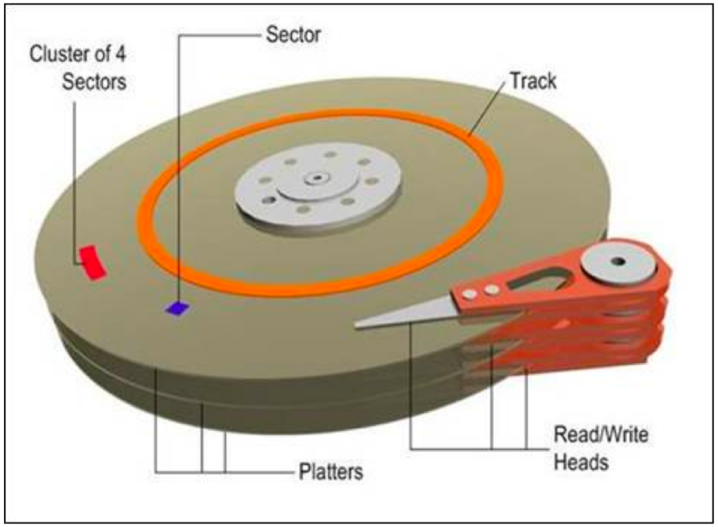
physical machine interface

hardware

  - Hide differences among different brands and interfaces
  - Minimize differences between similar types of devices

Physical Disk Structure

- Disk components
  - Platters
  - Tracks
  - Sectors
  - Heads



Disk Performance

- What does disk performance depend upon?
  - Queue – wait for the disk to be free
  - Positioning – move the disk arm to the correct track and rotate to the right sector (seek & rotate)
  - Access – transfer data from/to disk

- For given load, performance depends on
  - Positioning overhead (~ 1-10ms)
  - Transfer time (~ 100MBps)

Optimizing I/O performance

- To increase performance of slow I/O devices:
  - Avoid doing I/O
  - Reduce overhead
  - Amortize overhead over larger request

- Efficiency = transfer time / (positioning time + transfer time)
  - Rule of thumb: Achieve at least 50% efficiency
  - Example: 5ms avg. seek time and 100MBps transfer rate → Read at least 500KB

Disk scheduling

- Reduce overhead by reordering requests
  - Can be implemented in OS or hardware (Tradeoffs?)
- Examples:
  - FCFS (first come, first served)
  - SSTF (shortest seek time first)
  - SCAN (sort requests by position)

How else can OS reduce overhead in disk I/O?

  - SSTF can lead to starvation, longer track travel distances
- Does CPU scheduling policy affect throughput?
  - Not much for workloads where I/O matters
- What about queuing delays?
  - Better scheduling also drains the queue faster

Optimizing data layout

- Keep related items together on disk
- How to know what items will be accessed together?
  - Based on general usage patterns
  - Based on past accesses of data

Flash RAM

- Optimizations depend on specifics of a device
- Flash RAM has different characteristics than magnetic disk
  - Better read performance (but still slow writes)
    - » Random read: 25μs, Sequential read: 30ns
  - Lower power
  - Better shock resistance
  - But also has some issues: wearout, no overwrite
- OS hides physical characteristics of device from applications

Optimizing I/O performance with Flash RAM

- Move data blocks to do wear leveling
- Write data in big blocks
- Prefer to read data rather than write
  - Caching is important

File systems

- File system: a data structure stored on a persistent medium
- Ensures that data persists across …
  - Power outages
  - Machine crashes/reboots
  - Process births/deaths
- How to enable persistence across these events?
  - Use persistent storage medium
  - Write data carefully
  - Avoid use of addresses that change across processes

Interface to file system

- Create file
- Delete file
- Read <file, offset>
- Write <file, offset>
- Other (e.g., list files in a directory)

- Alternate interface?
  - SQL → Database

File system workloads

- Optimize data structure for the common case

- Some general rules of thumb
  - Most file accesses are reads
  - Most programs access files sequentially and entirely
  - Most files are small, but most bytes belong to large files

File abstraction

- Reality: One (or a few) disks to store data
- Abstraction: Numerous storage objects (files)
- Challenges:
  - How to name files?
  - How to find and organize files?

How to store a file?

- Need to store metadata
  - File size
  - Owner/Permissions
  - Time of creation/last access
- Need to store pointer to data
  - Pointer must be independent of process
- Basic data structure: a file header
  - inode in Unix, Master File Table record in NTFS
  - Structure that describes file and allows you to find data

Administrivia

- Handing back regrade requests
- Project 3 due in 9 days
- Remember to spread the commits

Contiguous allocation

- File = array of blocks ("extent")
  - Reserve space in advance
  - If file grows, move it to a larger free area
  - File header contains starting location of file and size

- Pros and cons?

  + Fast sequential access

  + Easy random access

  - Wastes space; external fragmentation

  - Difficult to grow file

Indexed files

- File = array of block pointers
  - Just like page table

File block #	Disk block #
0	18
1	50
2	8
3	15

- Pros and cons?
  + Easy to grow file
  + Easy random access
  - But potentially slow for sequential access
- How to speed up sequential access?
  - To grow file, allocate new block close to previous block
  - “Close” could be same or nearby track/cylinder
  - Leave some free blocks to facilitate this

Indexed files

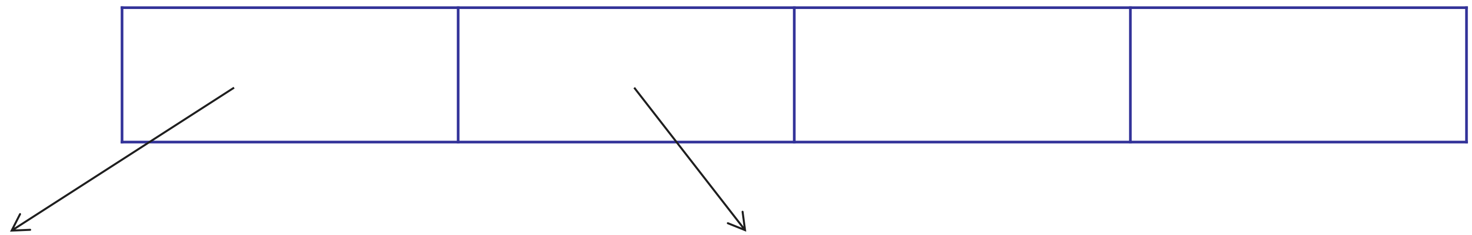
- Consequence of allowing for large files?
  - Large files are OK, small files are the problem
- Waste space in file header for small files
  - e.g. max file size = 16GB, file block = 4KB
  - 4M pointers in header →160GB of headers
- Solution: increase block size to 4MB?
  - Problem: internal fragmentation
- Trade-off between page table size and block size

File block #	Disk block #
0	18
1	50
2	8
3	15
…	
4194304	189

Multi-level indexed files

- File = tree of block pointers

level 1
node



File block #	Disk block #
0	18
1	50
2	8
3	15

File block #	Disk block #
4	20
5	11
6	3
7	43

**level 2
node**

- Pros?
  - Files can easily grow, appending to files is easy
  - Allows large file, but small files don’t waste header space

Multi-level indexed files

- Downsides?
  - Could have lots of seeks for sequential access

  → Bad performance, especially for large files
- How to fix?
  - Caching
  - Non-uniform depth