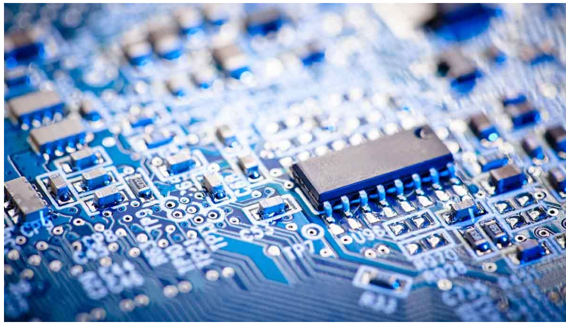
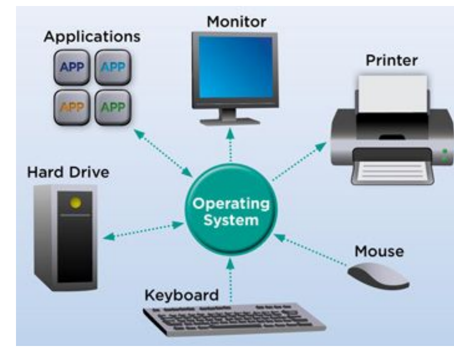
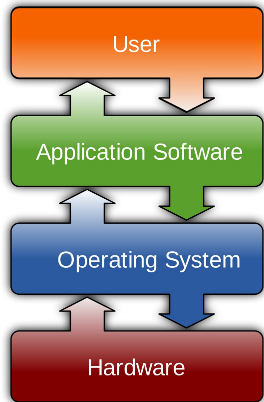
Please do not redistribute these slides without prior written permission





CS 3650 Computer Systems

Drs. Alden Jackson / Ferdinand Vesely



Intro	Virtualization		Concurrency	Persistence	Appendices
Preface	3 *Dialogue*	12 *Dialogue*	25 *Dialogue*	35 *Dialogue*	*Dialogue*
TOC	4 Processes	13 Address Spaces	26 Concurrency and Threads code	36 I/O Devices	Virtual Machines
1 *Dialogue*	5 Process API code	14 Memory API	27 Thread API	37 Hard Disk Drives	*Dialogue*
2 Introduction code	6 Direct Execution	15 Address Translation	28 Locks	38 Redundant Disk Arrays (RAID)	Monitors
	7 CPU Scheduling	16 Segmentation	29 Locked Data Structures	39 Files and Directories	*Dialogue*
	8 Multi-level Feedback	17 Free Space Management	30 Condition Variables	40 File System Implementation	Lab Tutorial
	9 Lottery Scheduling code	18 Introduction to Paging	31 Semaphores	41 Fast File System (FFS)	Systems Labs
	10 Multi-CPU Scheduling	19 Translation Lookaside Buffers	32 Concurrency Bugs	42 FSCK and Journaling	xv6 Labs
	11 *Summary*	20 Advanced Page Tables	33 Event-based Concurrency	43 Log-structured File System (LFS)	
		21 Swapping: Mechanisms	34 *Summary*	44 Flash-based SSDs	
		22 Swapping: Policies		45 Data Integrity and Protection	
		23 Case Study: VAX/VMS		46 *Summary*	
		24 *Summary*		47 *Dialogue*	
				48 Distributed Systems	
				49 Network File System (NFS)	
				50 Andrew File System (AFS)	
				51 *Summary*	

Lecture 12 - File Systems

Persistence

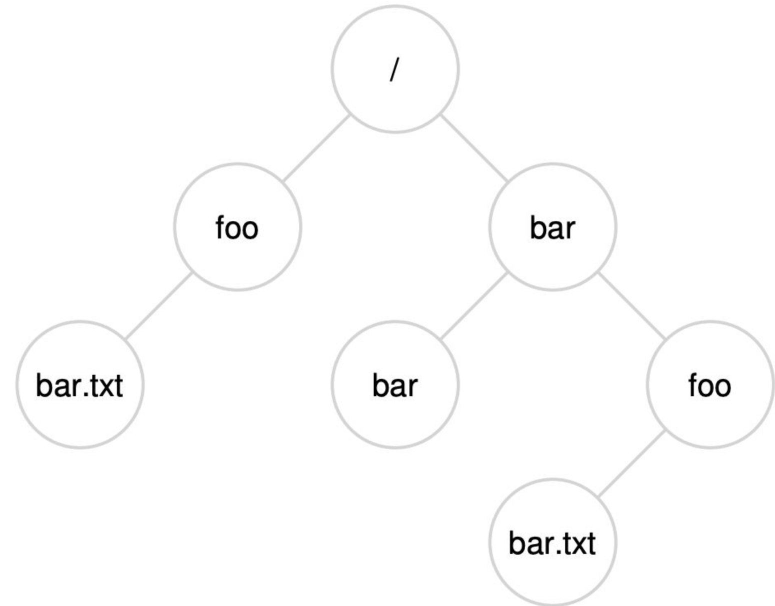
Introduction to File Systems

- Persistent storage devices allow storing data across reboots
  - non-volatile memory keeps data intact when the power goes away
- Data stored as a sequence of bits using some physical representation, e.g.,
  - magnetic (HDDs, floppy disks, tape drives)
  - optic (CDs)
  - electronic (SSDs)
- From a (low-level) software perspective:
  - data stored as a sequence of bytes
  - usually organized in blocks
- Long-term storage in a linear array would be a pain
  - Where to put a new file
  - External fragmentation: defrag/compaction is expensive
- How do we organize data?

Directory

- **Directory - a file** that has a *specific* structure
  - Like all files, it has an inode number

- The contents is a list of *(user-readable name, inode number)*
  - Each entry refers to a file or another directory, e.g.,
  - File “foo” with inode=10, (“foo”, “10”)
  - Directory tmp with inode=234, (“tmp”, “234”)
- Placing directories in other directories creates a tree



Organizing Data: Two Key Abstractions

- We recall a disk has some standardized contents, e.g., the MBR, the File System is another
- Modern file systems have two abstractions:

  a. **File** - a linear array of bytes, each of which can be read and written
  b. **Directory** - **a file** that has a specific structure

File

- File - a linear array of bytes, each of which can be read and written
  - Addressed using a low-level name, usually a number, called an **inode**
  - The text name is a helper for humans

- Most OSes don't know much about the structure of a file or its contents
  - Bytes are bytes
  - The job is store data persistently to give it back to the user when requested

UNIX directory conventions

- ‘.’ = cwd, i.e., itself
- ‘..’ = parent directory
- ‘/’ = root directory, inode=2 (special case)

File System API: Files

- The OS provides several system calls to interact with a file system
- We've worked with some file I/O operations:
  - `open` / `close` - open (or create) / close a file
  - `read` / `write` - read / write bytes from/to file
- There's more…

- `lseek` - move within a file
- `fsync` - flush a buffer to a file - force data to be written
- `rename` - rename a file
- `stat` - get information (metadata) about a file
- `link` - associate another name with file data
- `unlink` - remove (a reference to) a file / delete a file
- **Where is the file metadata?**

File System API: Directories

- `mkdir` - make directories
- `rmdir` - remove a directory (only if empty)
- `opendir` - open a directory stream
- `readdir` - read a directory entry from a directory stream
- `closedir` - close a directory stream

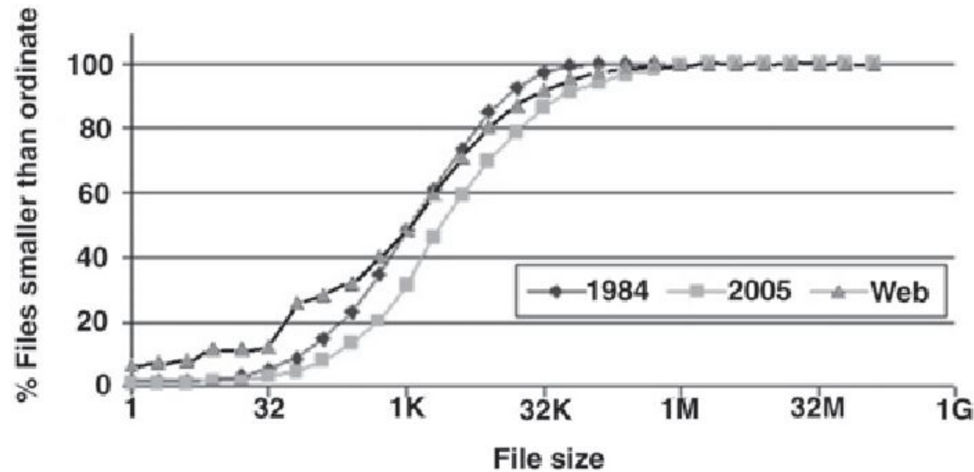
File System Implementation

- File systems are implemented as “drivers”
  - but they do not abstract the hardware directly
- They provide a software abstraction over the lower-level storage APIs

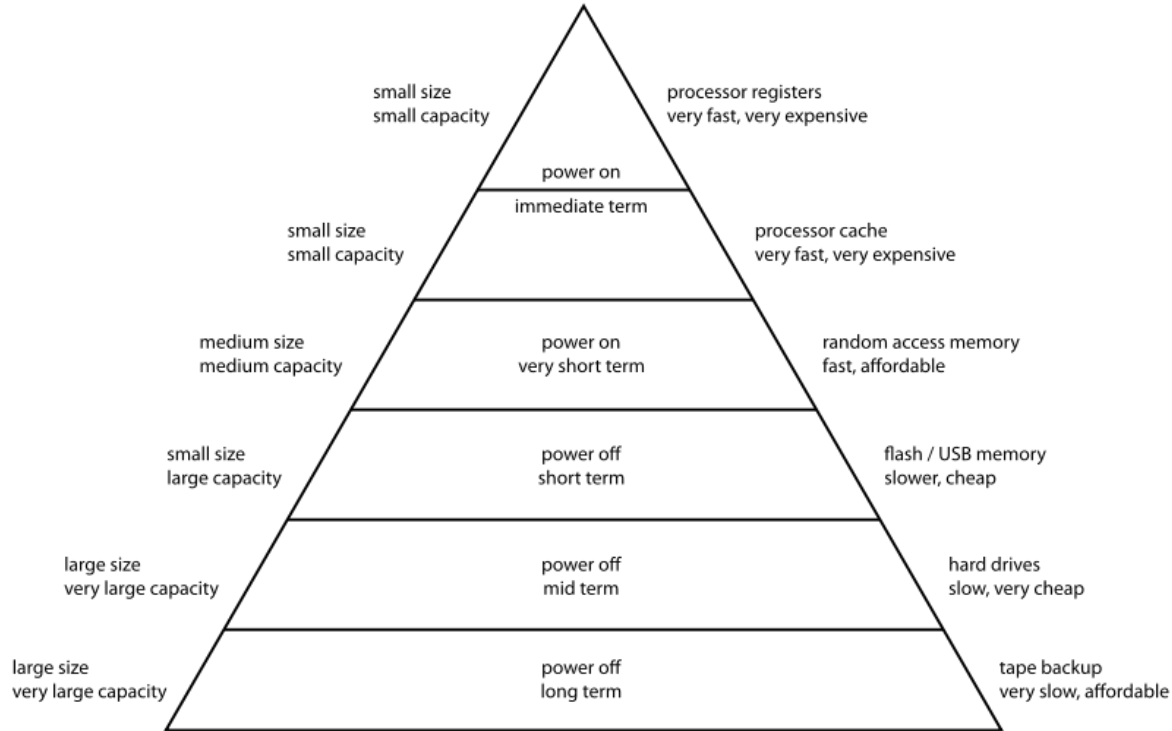
- All they really do:
  - Translate a name + offset to the appropriate disk position
  - Keep track of where to store data (free space management)

File System Implementation

- To design a FS, we really need to understand and decide on two things:
  - Data structures
  - Access methods
- Data structs - how is data and metadata organized
  - arrays, linked lists, trees, ...
- Access methods - how does the file API map to usage of these data structures
- We will look at a simple file system - related to the first Unix FS by Ken Thompson



From File size distribution on UNIX systems: then and now
A. Tanenbaum, J. Herder, H. Bos, ACM SIGOPS Oper. Syst. Rev., 2006
https://api.semanticscholar.org/CorpusID:4457775


Computer Memory Hierarchy
small size
small capacity
processor registers
very fast, very expensive
power on
immediate term
small size
small capacity
processor cache
very fast, very expensive
medium size
medium capacity
power on
very short term
random access memory
fast, affordable
small size
large capacity
power off
short term
flash / USB memory
slower, cheap
large size
very large capacity
power off
mid term
hard drives
slow, very cheap
large size
very large capacity
power off
long term
tape backup
very slow, affordable

File system characteristics

- Most files are small
  - ~2K is the most common size
- The average file size is growing
  - ~200K
- Most bytes are in the large files
  - A few big file use up most of the space
- File systems contain lots of files
  - ~100K on average
- File systems are roughly half full
  - Even as disks grow the number of files is ~50%
- Directories are small
  - Many have few entries, most have 20 or less

Switch to other File System slides

The remaining slides for today's lecture were prepared by Prof Thomas Ropars at the University of Grenoble for his Operating Systems class.

They can be found on our website under Week 12.

They can be also be found on his course website at:
https://m1-mosig-os.gitlab.io/lectures/lecture_16--File_systems.pdf

Switch to other slides