CSE 4/521 Introduction to Operating Systems

Lecture 22 – File System Implementation I

(File-System Structure, File-System Implementation, Directory Implementation)

Summer 2018

Overview

Objective:

- To discuss the details of implementing local file systems and directory structures.

- File-System Structure
- File-System Implementation
- Directory Implementation

Recap

- RAID Structure
  - RAID 0, RAID 1, RAID 2, RAID 3, RAID 4, RAID 5, RAID 6, RAID (0+1), and RAID (1+0)
- Stable-Storage Implementation
  - Write-ahead log scheme
  - Maintain 2 physical block per logical block

Questions

1. What is RAID and why is it used? (Easy)

2. Why do some RAID level implement block striping rather than bit striping? (Easy)

3. Compare the performance of write operations on RAID level 5 compared to RAID level 1. (Medium)

4. Even if RAID 0 is worse, why would I use it anyway? (Easy)

Overview

- File-System Structure
- File-System Implementation
- Directory Implementation

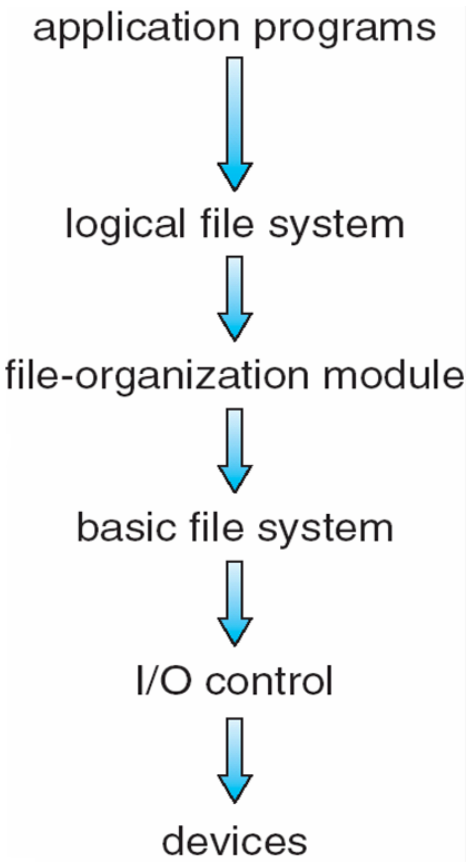
Overview

- File-System Structure
- File-System Implementation
- Directory Implementation

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk
- Disk provides in-place rewrite and random access
  - I/O transfers performed in blocks of sectors (usually 512 bytes)
- File control block – storage structure consisting of information about a file
- Device driver controls the physical device
- File system organized into layers

File-System Structure: Layered File System



File-System Structure: File System Layers

- Device drivers manage I/O devices at the I/O control layer
  - Given commands like "*read drive1, cylinder 72, track 2, sector 10, into memory location 1060*" outputs low-level hardware specific commands to hardware controller
- Basic file system given command like "*retrieve block 123*" translates to device driver
  - Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- File organization module understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation

File-System Structure: File System Layers

- Logical file system manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
- Logical layers can be implemented by any coding method according to OS designer

File-System Structure: File System Layers

- Many file systems, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660; Unix has UFS, FFS; Windows has FAT, FAT32, NTFS as well as CD, DVD Blu-ray, Linux has more than 40 types, with extended file system ext2 and ext3 leading; plus distributed file systems, etc.)
  - New ones still arriving – ZFS, GoogleFS, Oracle ASM, FUSE

Overview

- File-System Structure
- File-System Implementation
- Directory Implementation

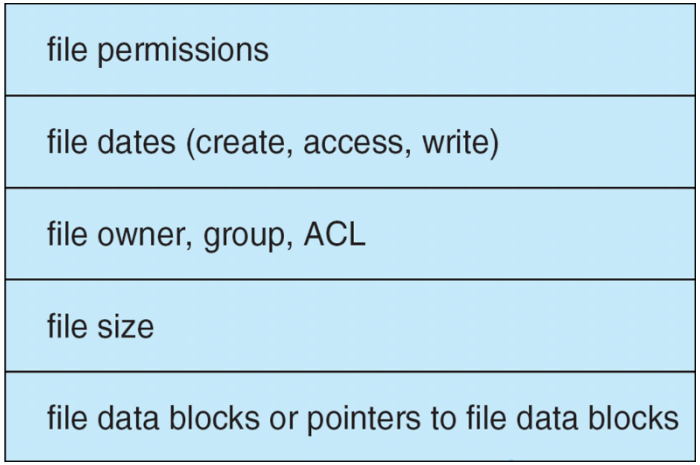
File-System Implementation

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures

1. Boot control block contains info needed by system to boot OS from that volume
   - Needed if volume contains OS, usually first block of volume
2. Volume control block (superblock, master file table) contains volume details
   - Total # of blocks, # of free blocks, block size, free block pointers or array
3. Directory structure organizes the files
   - Names and inode numbers, master file table

File-System Implementation

4. Per-file File Control Block (FCB) contains many details about the file
   - inode number, permissions, size, dates
   - NFTS stores into in master file table using relational DB structures

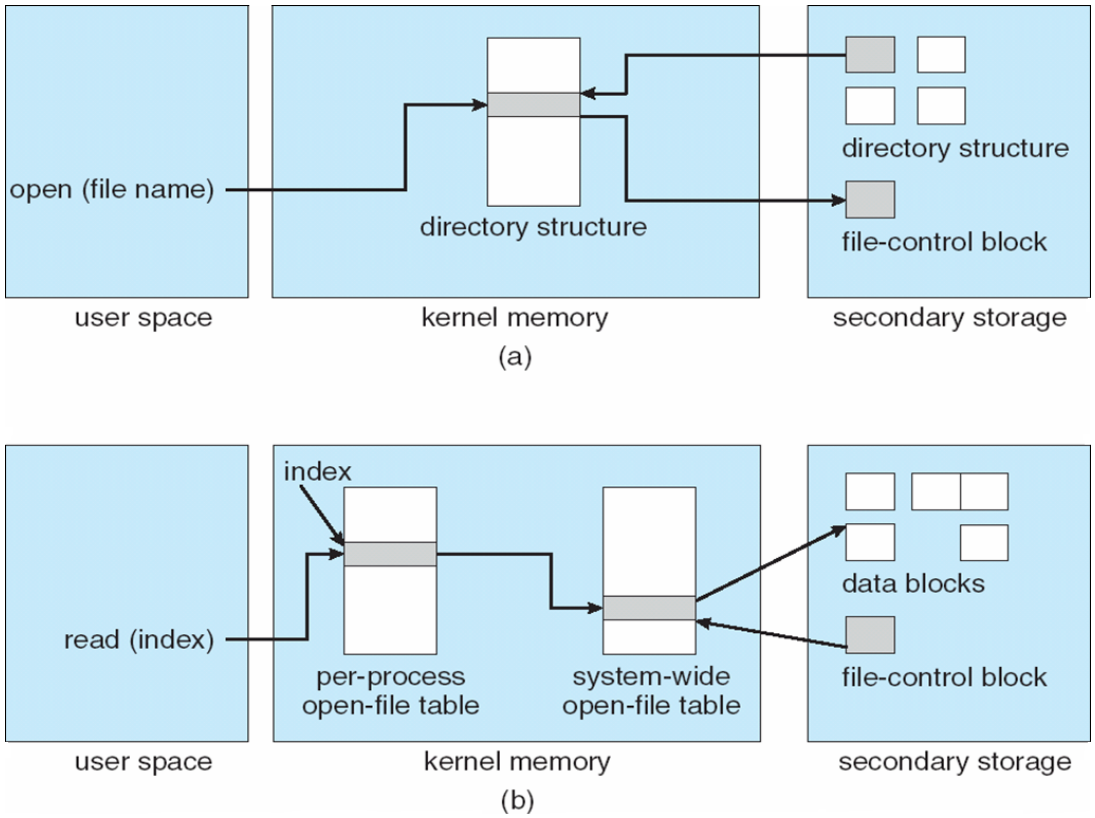


File-System Implementation: In-Memory File System Structures

1. Mount table storing file system mounts, mount points, file system types
2. In-memory directory structure.
3. Per-process open-file table.
4. System-wide open-file table.

- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure 12-3(a) refers to opening a file
- Figure 12-3(b) refers to reading a file
- Buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address

File-System Implementation: In-Memory File System Structures



File-System Implementation: Partitions and Mounting

- Partition can be a volume containing a file system ("cooked") or raw – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
- Root partition contains the OS, other partitions can hold other Oses, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - If yes, add to mount table, allow access

Overview

- File-System Structure
- File-System Implementation
- Directory Implementation

Directory Implementation

- Linear list of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- Hash Table – linear list with hash data structure
  - Decreases directory search time
  - Collisions – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Credits for slides

Silberschatz, Galvin and Gagne