



File System Implementation



- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- NFS
- Example: WAFL File System



Objectives



- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs



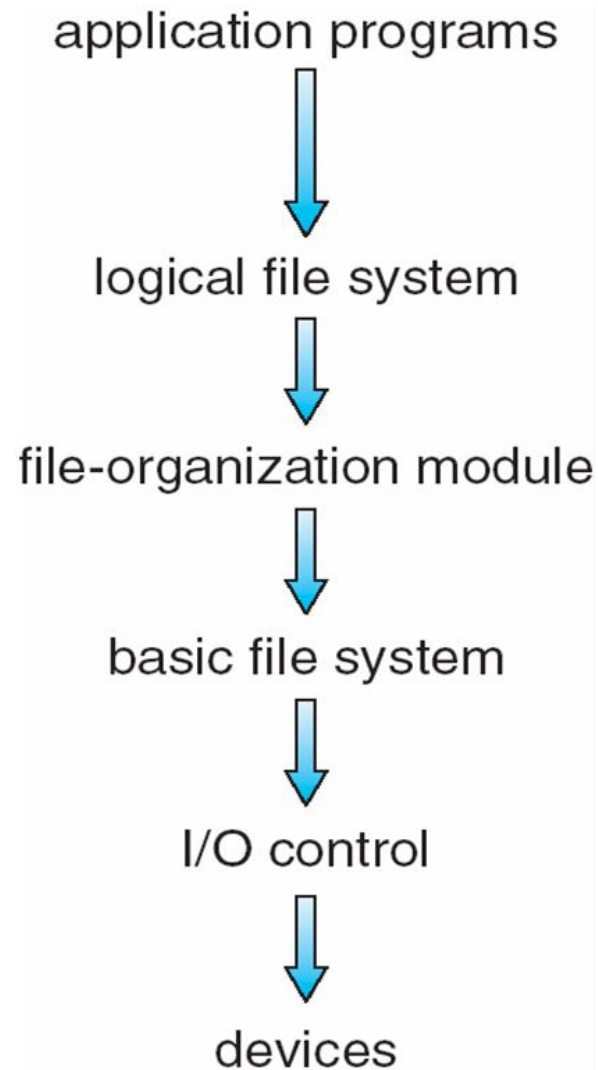
File-System Structure



- 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 by allowing data to be stored, located retrieved easily
- 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



Layered File System





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 Layers (Cont.)



- **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, Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
 - Logical layers can be implemented by any coding method according to OS designer



File System Layers (Cont.)



- 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 floppy, 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



File-System Implementation



- We have system calls at the API level, but how do we implement their functions?
 - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
 - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
 - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
 - Names and inode numbers, master file table



File-System Implementation (Cont.)



- 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 permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks



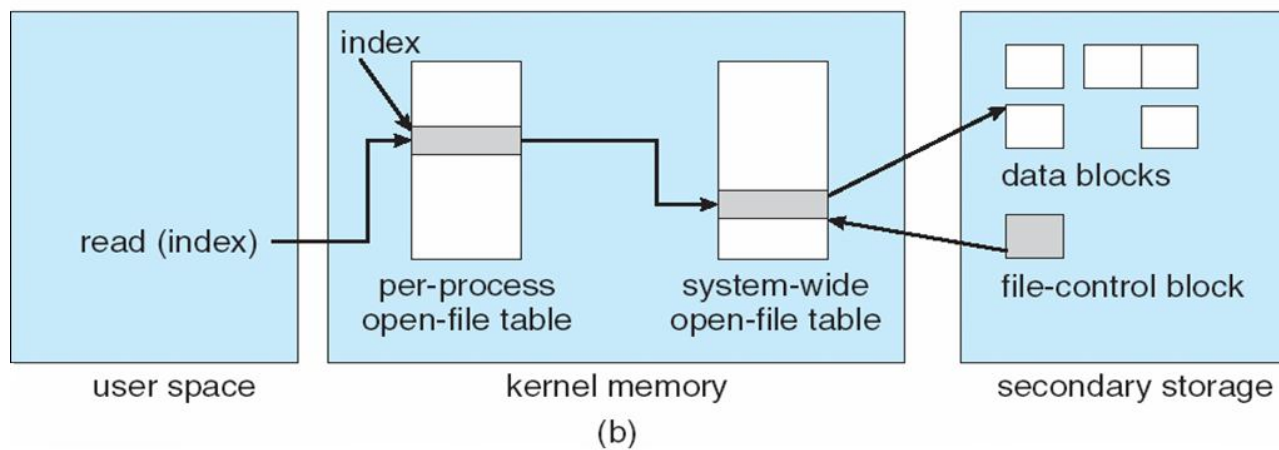
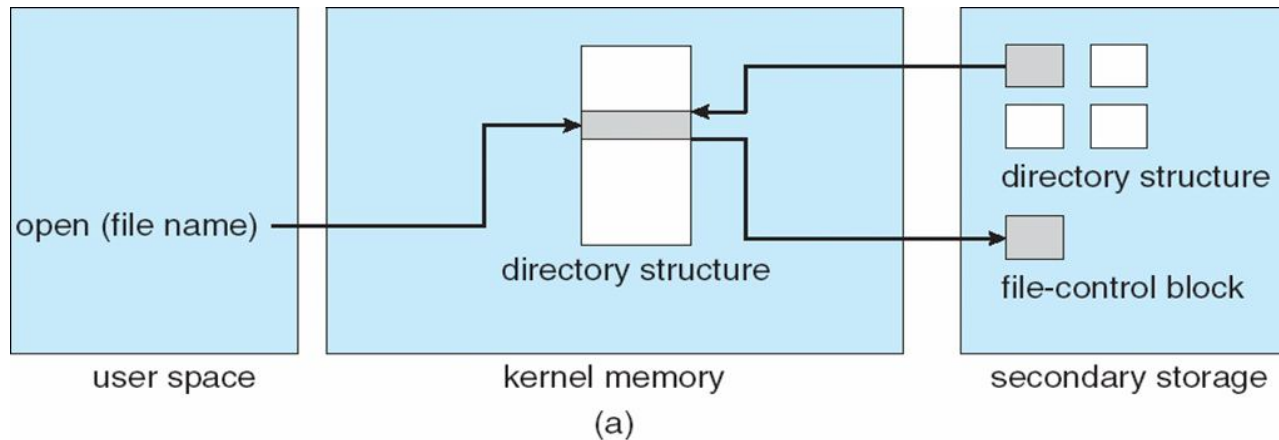
In-Memory File System Structures



- Mount table storing file system mounts, mount points, file system types
- 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
- Plus 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



In-Memory File System Structures





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
 - Or a boot management program for multi-os booting
- **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



Virtual File Systems



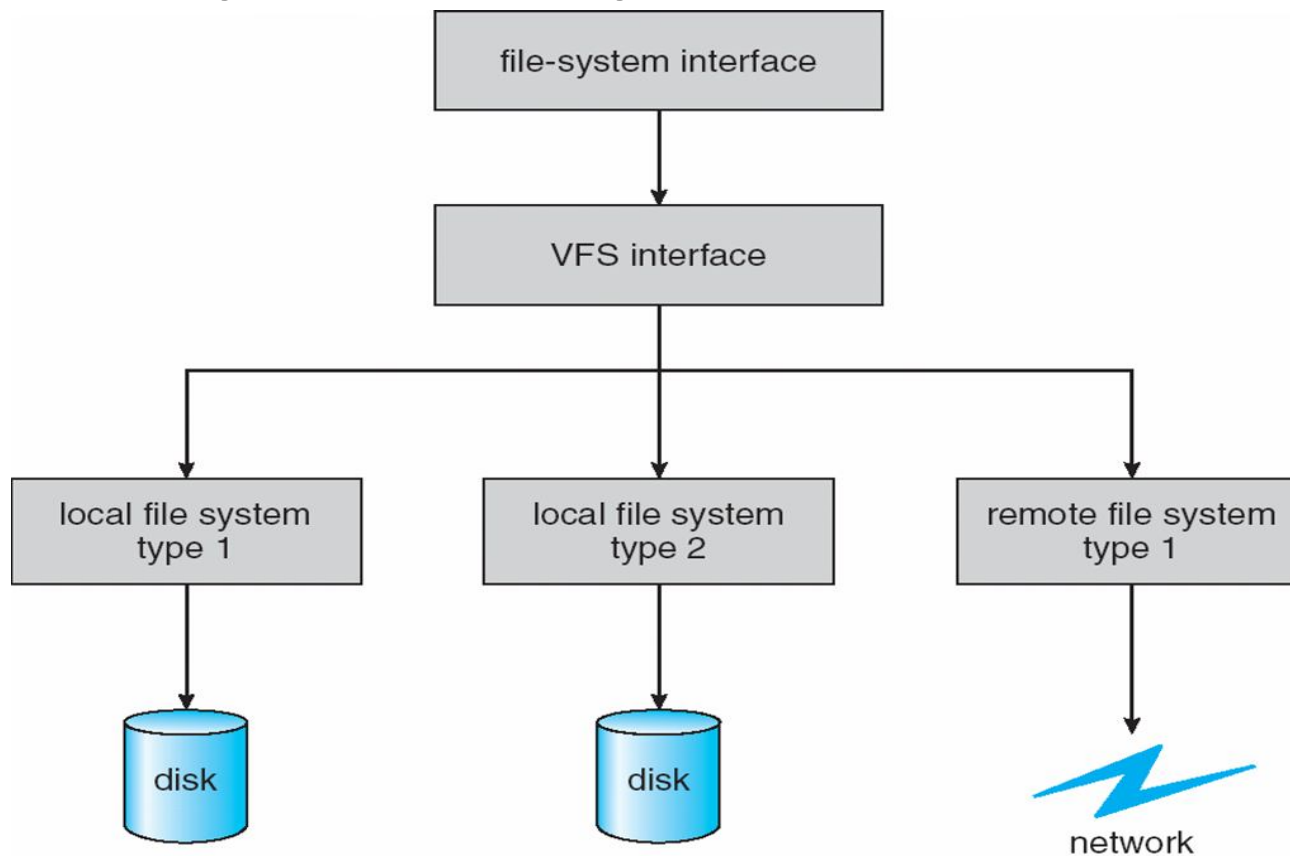
- **Virtual File Systems (VFS)** on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
 - Separates file-system generic operations from implementation details
 - Implementation can be one of many file systems types, or network file system
 - Implements **vnodes** which hold inodes or network file details
 - Then dispatches operation to appropriate file system implementation routines



Virtual File Systems (Cont.)



- The API is to the VFS interface, rather than any specific type of file system





Virtual File System Implementation



- For example, Linux has four object types:
 - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
 - Every object has a pointer to a function table
 - Function table has addresses of routines to implement that function on that object
 - For example:
 - `int open(. . .)`—Open a file
 - `int close(. . .)`—Close an already-open file
 - `ssize_t read(. . .)`—Read from a file
 - `ssize_t write(. . .)`—Write to a file
 - `int mmap(. . .)`—Memory-map a file



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