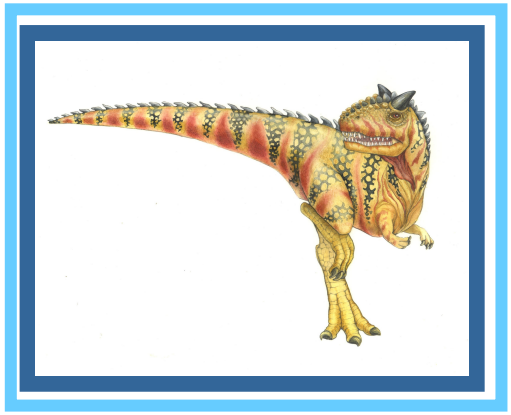
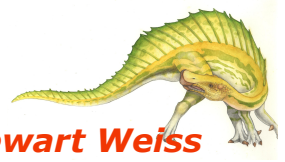
Chapter 11: File System Implementation



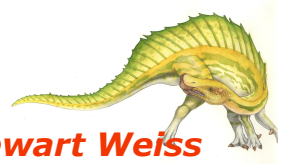
Chapter 11: File System Implementation

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Log-Structured File Systems



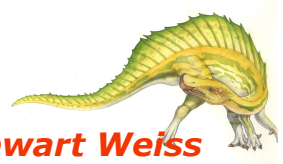
Objectives

- 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

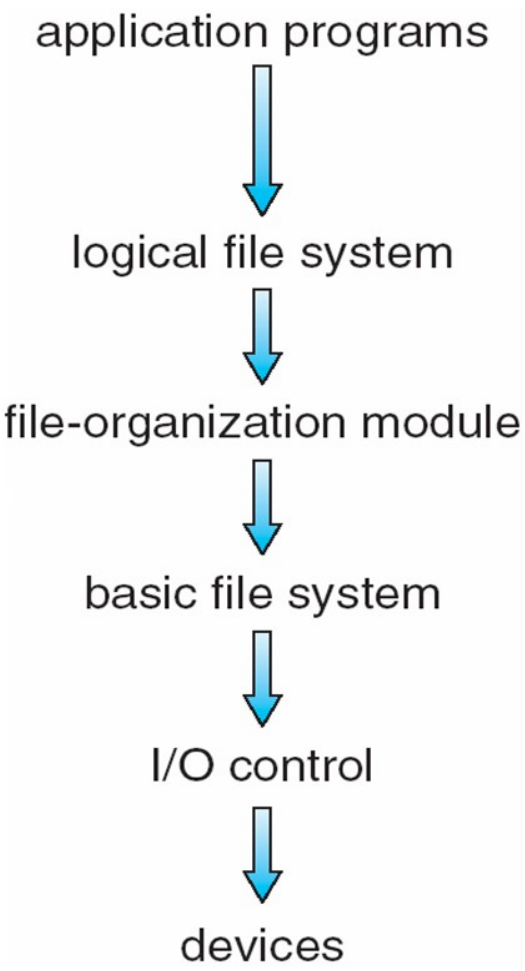


File-System Structure

- File system resides on disks
  - Basic unit is block
  - Random access to blocks
  - I/O transfers are in units of blocks
  - Block maybe spread across multiple sectors
  - Blocks have addresses: drive #, cylinder #, track #, sector #
  - Blocks have logical addresses as well – 0, 1,, 3, ...
- File system organized into layers – see next slide
  - Reduces code redundancy since multiple file systems can share lower levels (e.g. drivers, basic file system)
  - Increases system overhead because of excess calls
- **File control block** – storage structure consisting of information about a file – called inode in UNIX



Layered File System

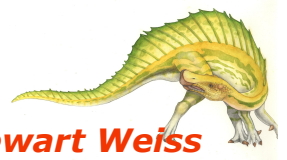


directory structure, file attributes, logical block assignment

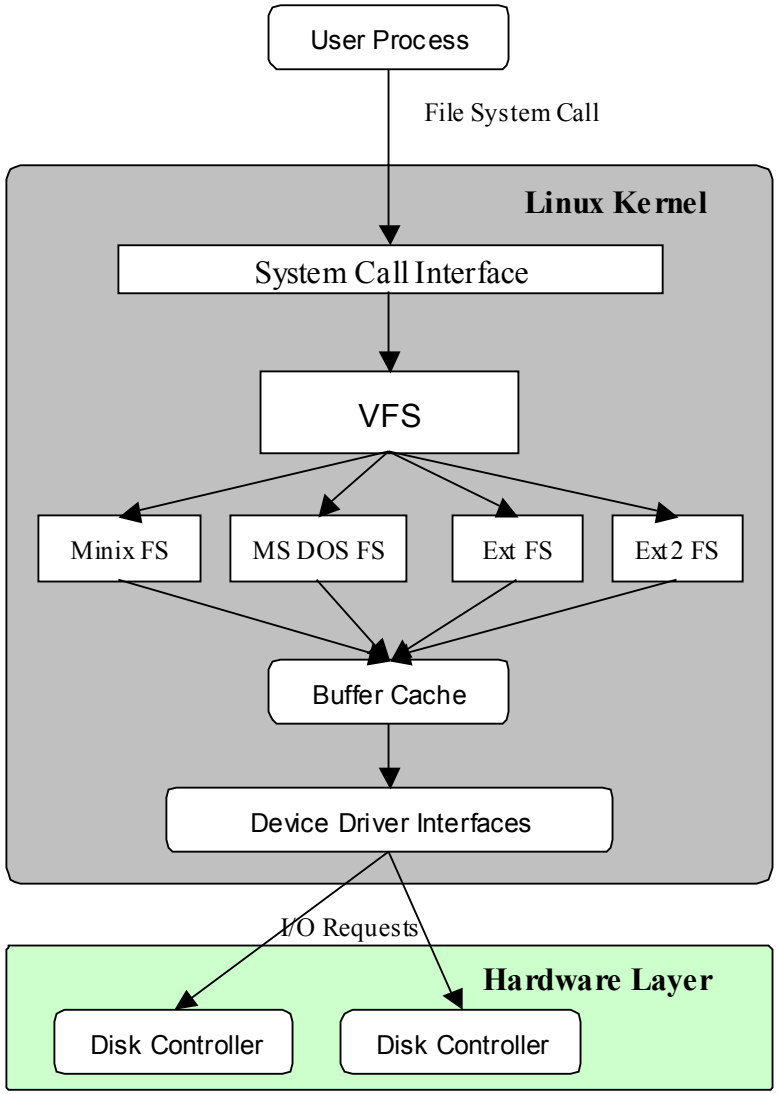
maps logical blocks to physical blocks, manages free space and allocation of blocks

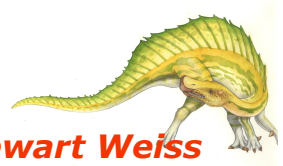
manages memory buffers and disk cache, issues commands to drivers

device drivers, issue commands to device controller to read/write blocks



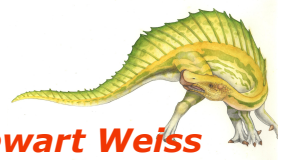
Linux Ext2 File System





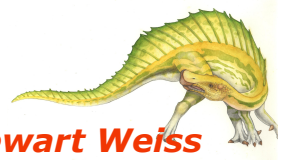
A Typical File Control Block

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



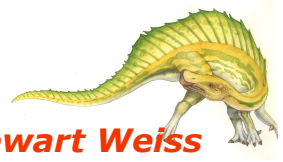
File Systems in Use

- non-Linux UNIX – UNIX File System (UFS), Berkeley Fast File System (FFS),
- Windows – FAT, FAT32, NTFS
- Linux – EXT2, EXT3, and > 40 different systems in general
- All support ISO-9660 (CD-ROMs) and DVD and floppy formats.



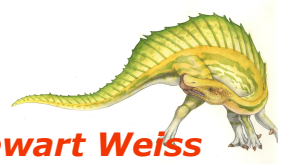
File System Implementation

- Basic structures include
  - ***Boot control block*** (one per volume) – data used by system for booting an operating system from that volume, if an OS is loaded there, otherwise empty. Usually first block on volume. Called ***boot block*** in UFS, ***partition boot sector*** in NTFS
  - ***Volume control block*** (one per volume) – contains number and sizes of blocks in volume, free-block count and list, free FCB count and list; called ***superblock*** in UFS; in ***master file table*** in NTFS.
  - ***Directory structure*** (one per file system) – collection of directories for organizing files – maps file names to FCBs (inodes in UNIX)
  - ***File control block*** (one per file) – attributes of file: size, location, owner, protections, timestamps, reference counts; called ***inode*** in UNIX.

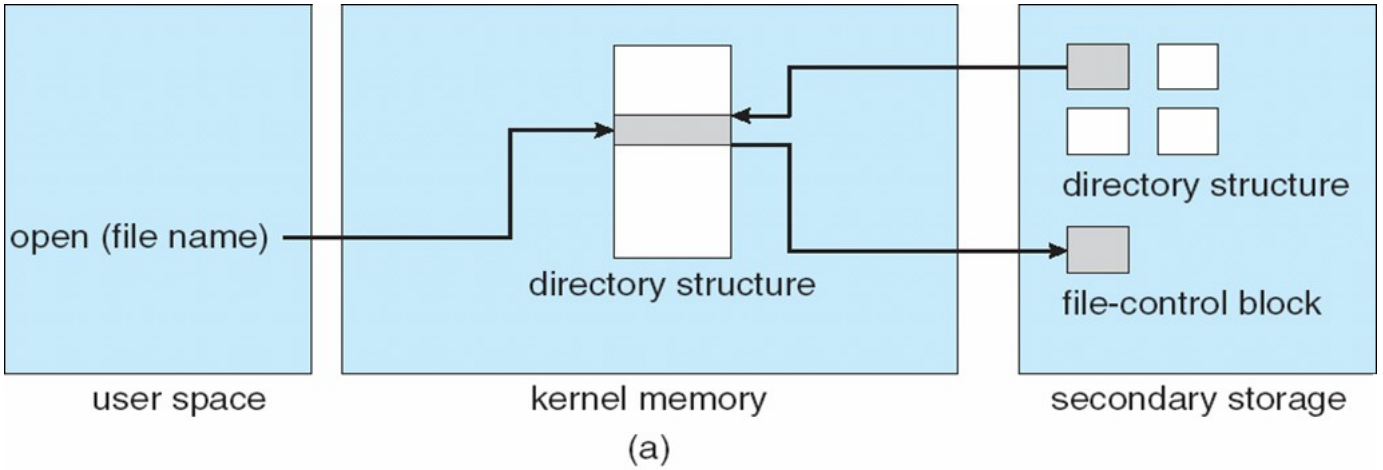


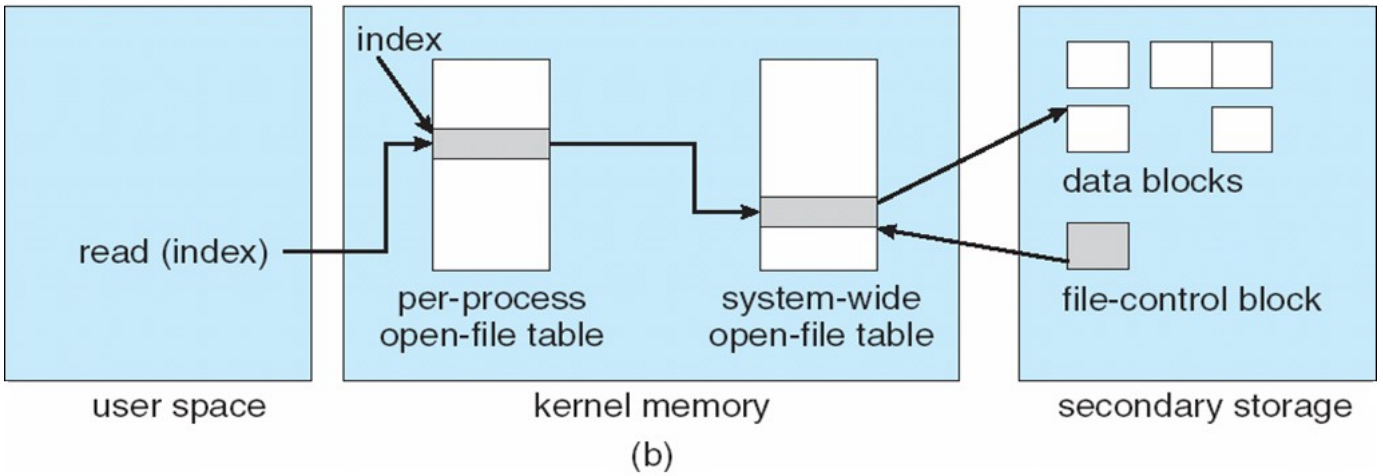
In-Memory File Structures

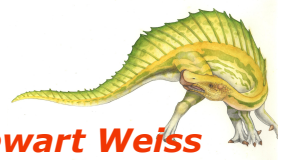
- The OS loads several structures into memory when a file system is mounted. These include
  - ***Mount table*** – contains information about the volume: access control, disk usage, block sizes, etc.;
  - ***Directory structure cache*** – a cache of recently accessed directories for faster performance;
  - ***System-wide open file table*** – copies of FCBs of every open file;
  - ***Per-process open file table*** – pointers to entries in the system wide open file table for files open by that process
  - Buffers for system blocks for reads/writes from/to disk
- Next slide shows structures used by open() and read() calls



In-Memory File System Structures

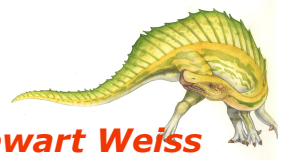






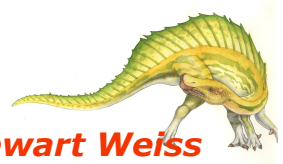
File Operations

- To create a file
  - Logical FS is called; allocates new FCB;
  - FCB is filled in with status
  - Data blocks are allocated
  - Address of data blocks filled in FCB
  - Directory in which new name is created is loaded into memory, updated
- To open a file:
  - New slot in per-process open file table created
  - OpSys uses file name to locate FCB (inode) for file;
  - creates new entry in system-wide open file table
  - if file already open, points new entry in system table to FCB in memory
  - if not, loads FCB into memory and points system entry to it
  - process open file table entry points to system table entry



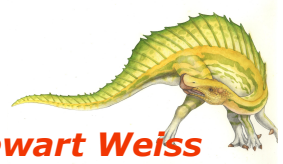
File Operations 2

- To close a file
  - Slot in per-process open file table is deleted
  - Reference count of entry in system-wide open file table is decremented
  - If reference count is zero, metadata is written back to disk and entry is deleted



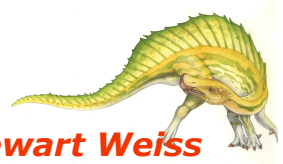
Partitions and Mounting

- A disk can be divided into multiple partitions
- Each partition can be either raw or have a file system
- Raw partitions are used by databases, for swap space,
- Boot information is stored usually in first few blocks of partition, with its own format, since OpSys is not yet loaded when it is read; the boot loader is stored there, read into memory and executed. The boot loader "knows" file system structure and can read OpSys into memory
- Root partition is the one containing the OpSys. It is mounted at boot time.
- Mounting verifies that device contains valid file system and updates system mount table with info about type of file system, mount point, access attributes, etc



Virtual File Systems

- Virtual File Systems (VFS) 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.
- The API is to the VFS interface, rather than any specific type of file system.



Schematic View of Virtual File System