OPERATING SYSTEMS

Storage Management

Venkatesh Prasad

Department of Computer Science

OPERATING SYSTEMS

Implementing File-Systems

Venkatesh Prasad

Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

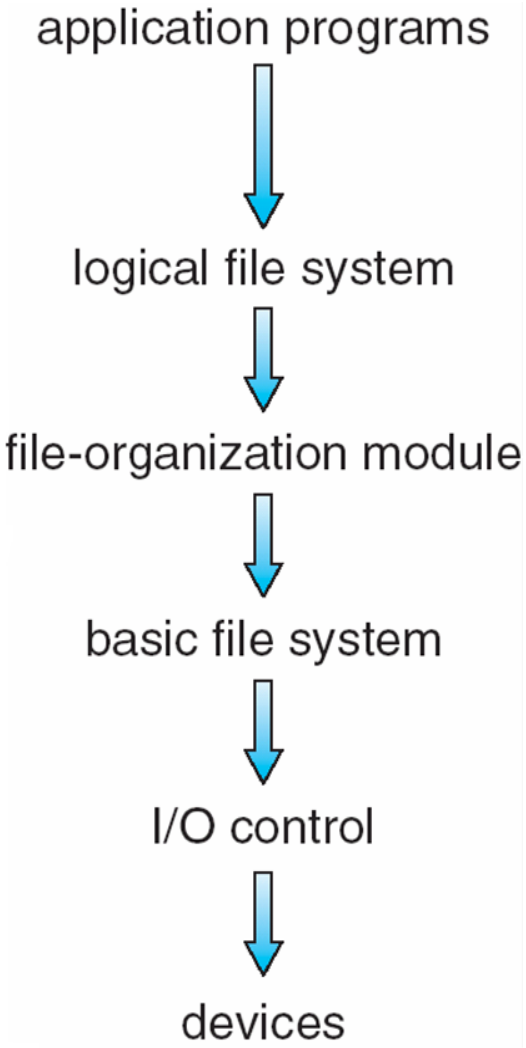
1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

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 (FCB)** – storage structure consisting of information about a file including ownership, permissions, and location of the file contents
- **Device driver** controls the physical device
- File system organized into layers

OPERATING SYSTEMS

Layered File System



Manages metadata information, and the dir structure to provide the file-org module. Also responsible for protection

Translates logical to physical block addresses

Issues generic commands to the device driver, manages mem buffers

Device drivers and interrupt handlers

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 System Layers (Cont.)

- **File organization module** understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation
- **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

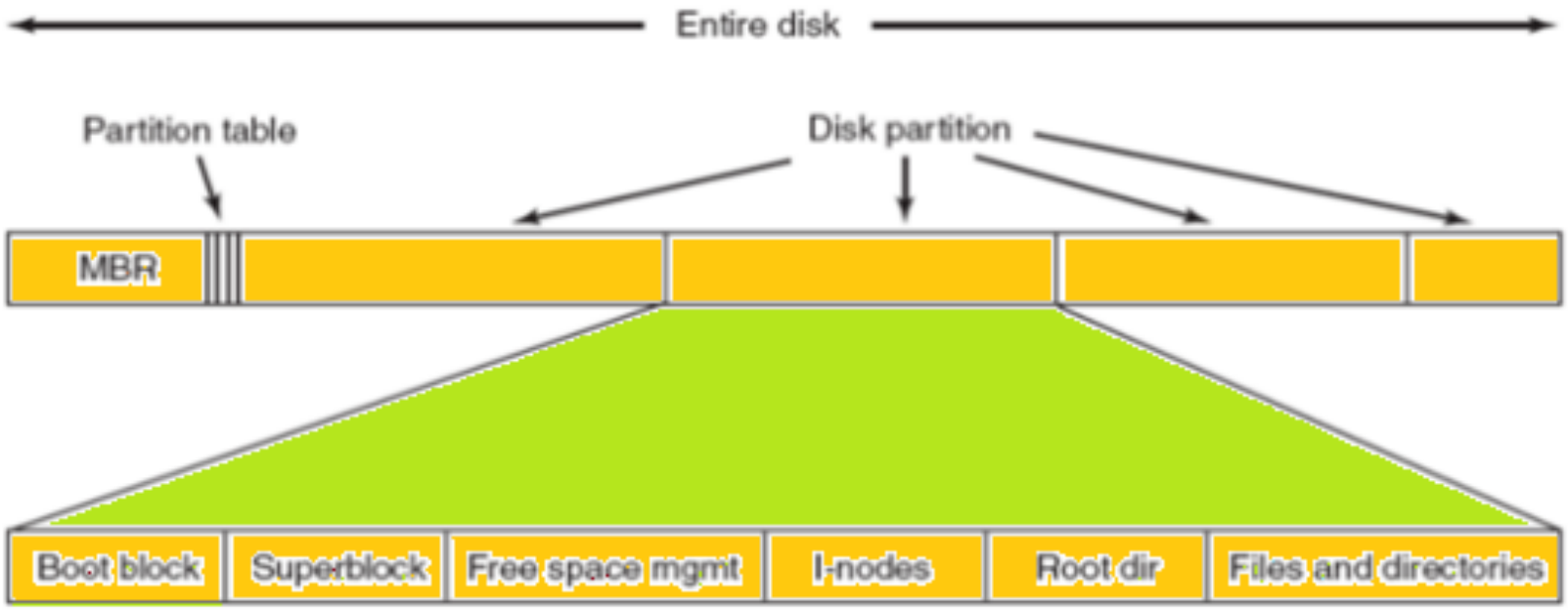
OPERATING SYSTEMS

File System Layers (Cont.)

- 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
- 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 – ZFS, GoogleFS, Oracle ASM, FUSE

OPERATING SYSTEMS

File-System Implementation



A possible file system layout.

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

OPERATING SYSTEMS

File-System Implementation (Cont.)

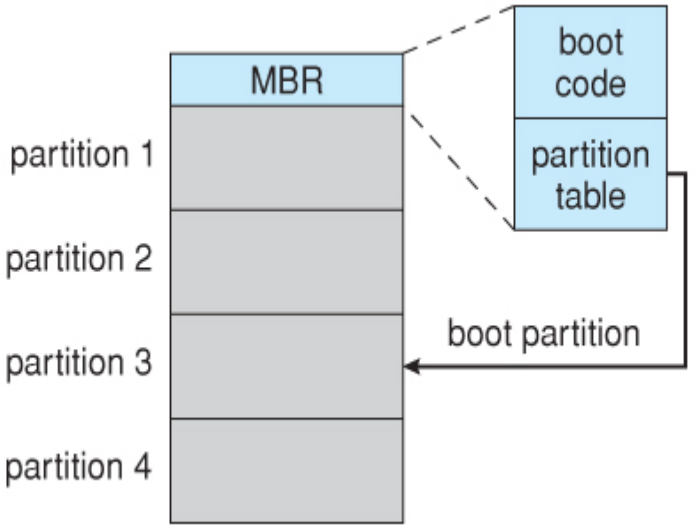
- Per-file **File Control Block (FCB)** contains many details about the file
  - inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures

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

OPERATING SYSTEMS

Boot Block

- Computer ROM contains a ***bootstrap*** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it
- ***Master Boot Record, MBR,*** is the first sector on the hard drive
  - contains a very small amount of code in addition to the ***partition table.***
  - The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the ***active*** or ***boot*** partition.
- The boot program then looks to the active partition to find an operating system
- In a ***dual-boot*** (or larger multi-boot) system, the user may be given a choice of which os to boot, with a default action to be taken in the event of no response within some time frame.
- Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS.
  - Kernel will start important services
  - Boot options at this stage may include ***single-user*** a.k.a. ***maintenance*** or ***safe*** modes, in which very few system services are started.



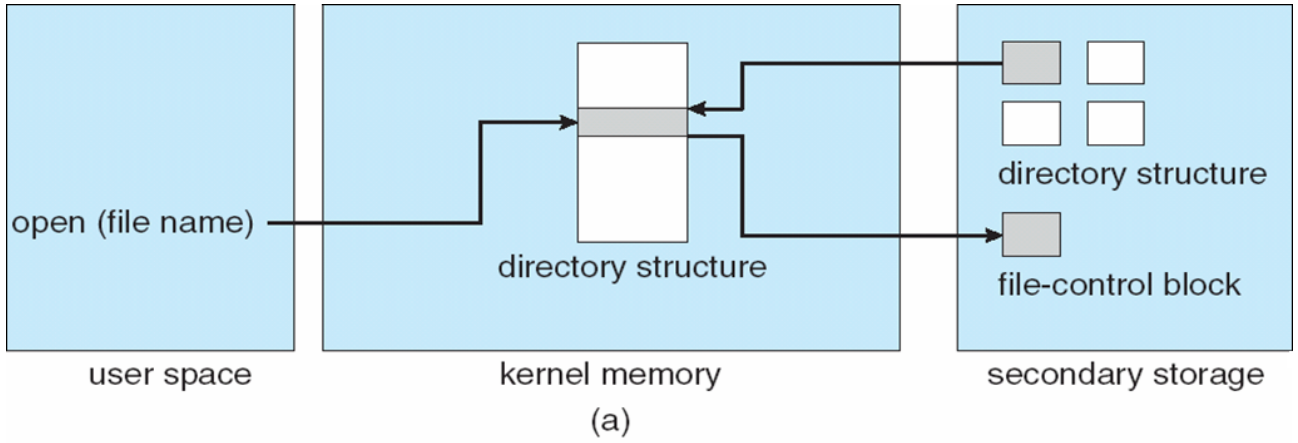
Booting from disk in Windows.

In-Memory File System Structures

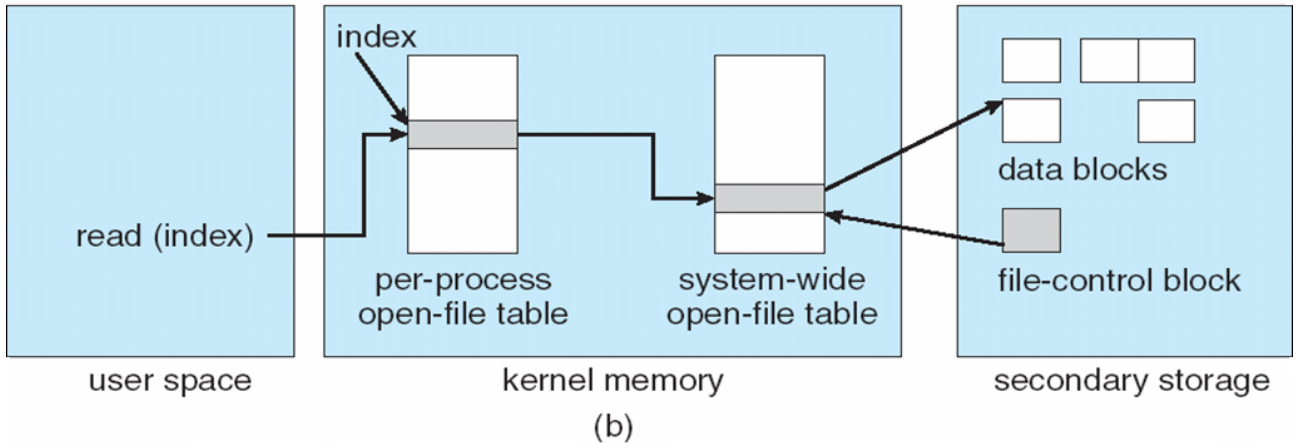
- Mount table storing file system mounts, mount points, file system types
- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure (a) refers to opening a file
- Figure (b) refers to reading a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address

OPERATING SYSTEMS

In-Memory File System Structures



(a)



(b)

OPERATING SYSTEMS

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 OS’s, 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

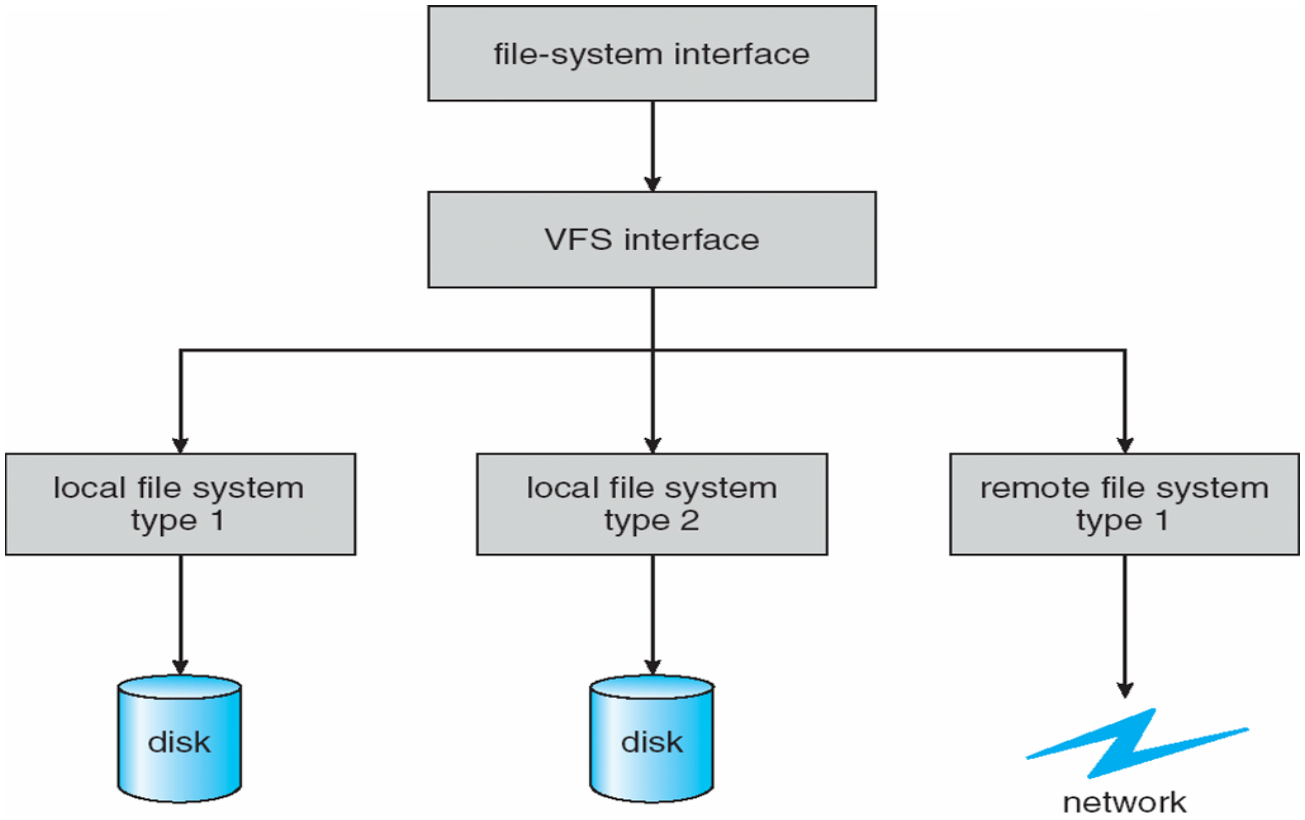
Virtual File Systems

- **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

OPERATING SYSTEMS

Virtual File Systems

- The API is to the VFS interface, rather than any specific type of file system (FS)
- VFS provides a single set of commands for the kernel and developers to access all types of FSs
- VFS software calls the specific device driver required to interface to various types of FSs
- Device driver interprets the standard set of FS commands to a specific type of FS on the partition or logical volume



Virtual File Systems

- For example, Linux has four object types:
  - **Inode object, file object, superblock object, dentry object**
- 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

THANK YOU

Venkatesh Prasad
Department of Computer Science Engineering

venkateshprasad@pes.edu