OPERATING SYSTEMS

Storage Management - 8

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Storage Management - 7:
File System Implementation

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 3

Unit 4: Storage Management

Mass-Storage Structur - Mass-Storage overview, Disk Scheduling, Swap-Space Management, RAID structure. File System Interface - file organization/structure and access methods, directories, sharing File System Implementation/Internals: File control Block (inode), partitions & mounting, Allocation methods.

Case Study: Linux/Windows File Systems

OPERATING SYSTEMS

Course Outline

37	Mass-Storage Structure: Mass-Storage overview	12.1	82.1
38	Disk Scheduling – FCFS, SSTF, SCAN, C-SCAN, LOOK	12.4	
39	Swap-Space Management, RAID Structure	12.6,12.7	
40	File Concept, File Structure, Access Methods	10.1-10.2	
41	Directory and Disk Structure	10.3	
42	File-System Mounting, File Sharing, Protecting	10.4-10.6	
43	Implementing File-Systems: File control Block (inode), partitions & mounting	11.1,11.2	
44	Disk Space Allocation methods: Contiguous, Linked, Indexed	11.4	
45	Case Study: Unix/Linux File systems	11.8	
46	NFS	16.7	

Topic Outline

- File-System Structure
- File-System Implementation
- Virtual File Systems
- Virtual File Systems Implementation

File System Structure

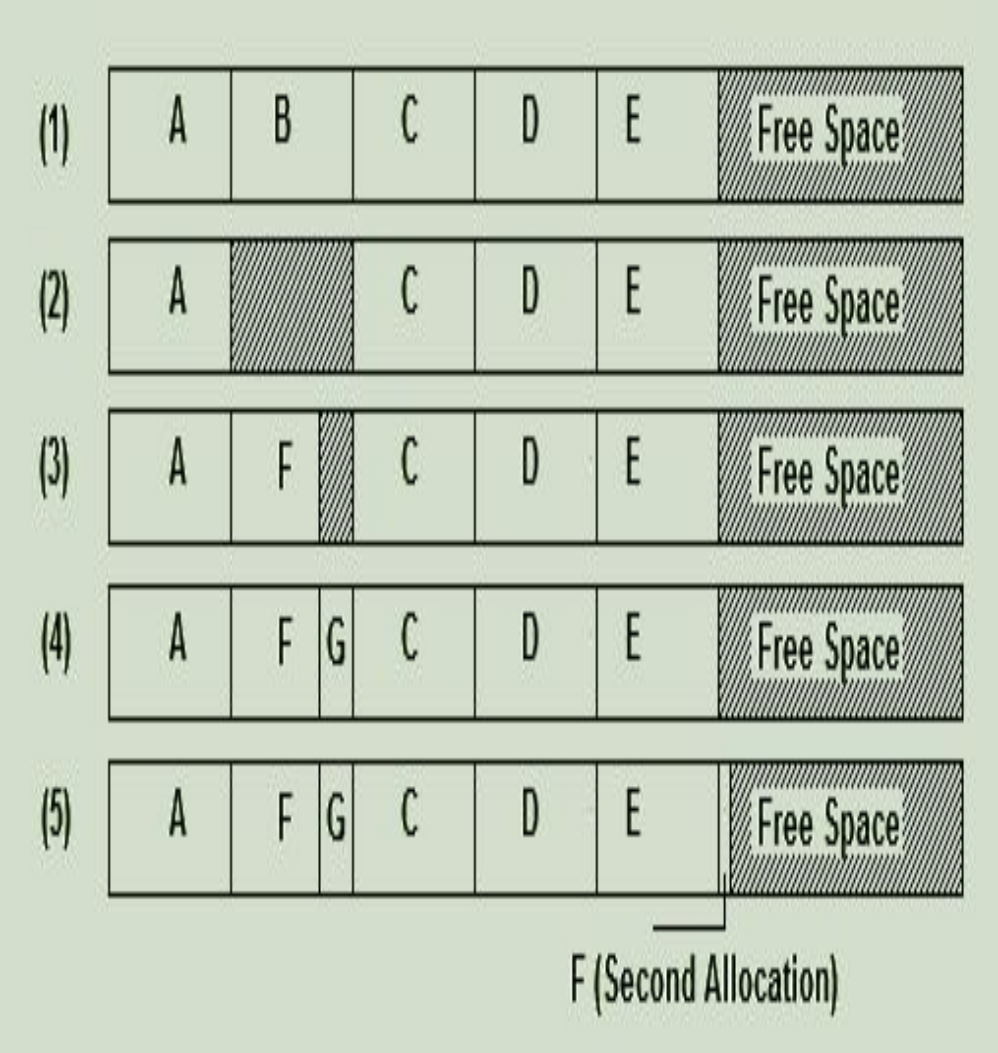
- File structure
  - Logical storage unit
  - Collection of related information

- **File system** resides on secondary storage (disks)

  - Provides user interface to storage, mapping logical to physical

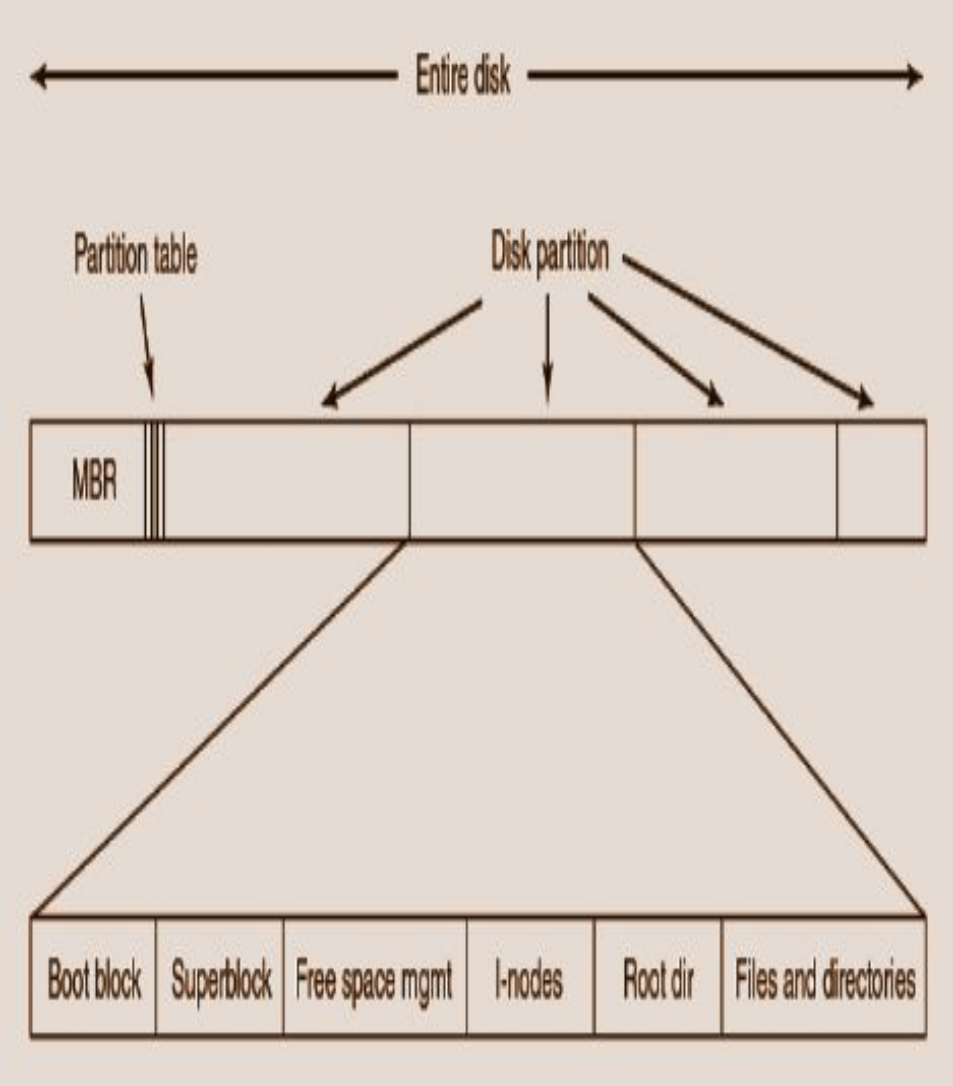
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily

File System Structure



- File system implementation defines how files and directories are stored, how disk space is managed, and how to make everything work efficiently and reliably.

File System Structure

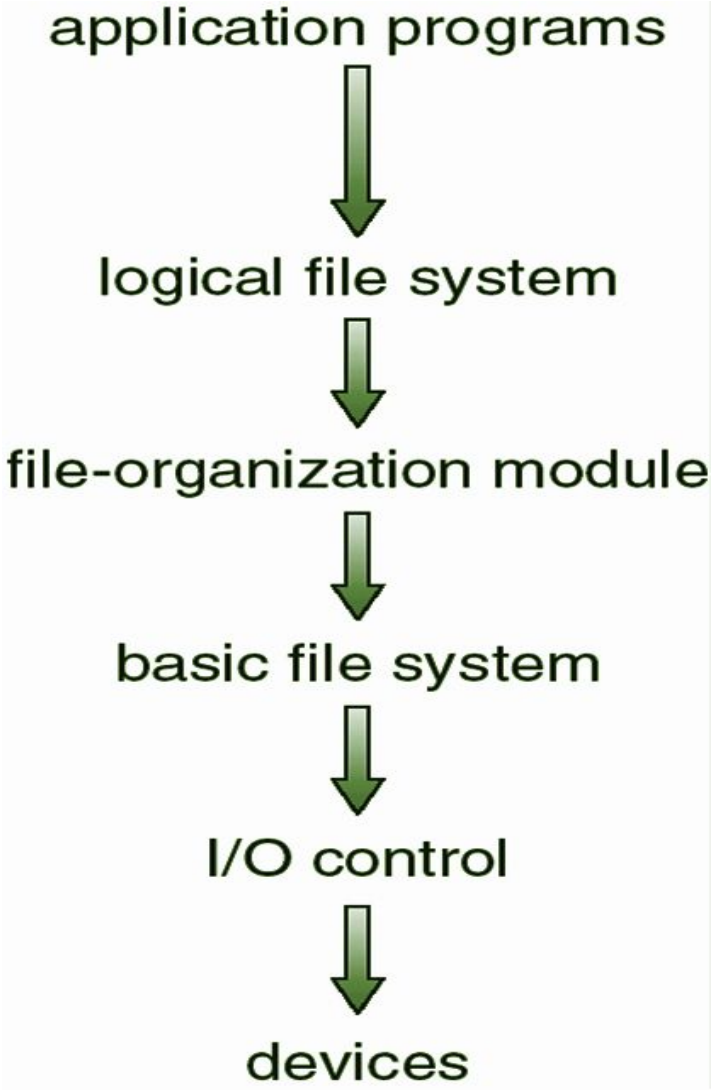


- File Systems are stored on disks. The figure depicts a possible File-System Layout.
- **MBR**: Master Boot Record is used to boot the computer
- **Partition Table**: Partition table is present at the end of MBR. This table gives the starting and ending addresses of each partition.
- **Boot Block**: When the computer is booted, the BIOS reads in and executes the MBR. The first thing the MBR program does is locate the active partition,read in its first block, which is called the boot block, and execute it. The program in the boot block loads the operating system contained in that partition. Every partition contains a boot block at the beginning though it does not contain a bootable operating system.
- **Super Block**: It contains all the key parameters about the file system and is read into memory when the computer is booted or the file system is first touched.

File System Structure

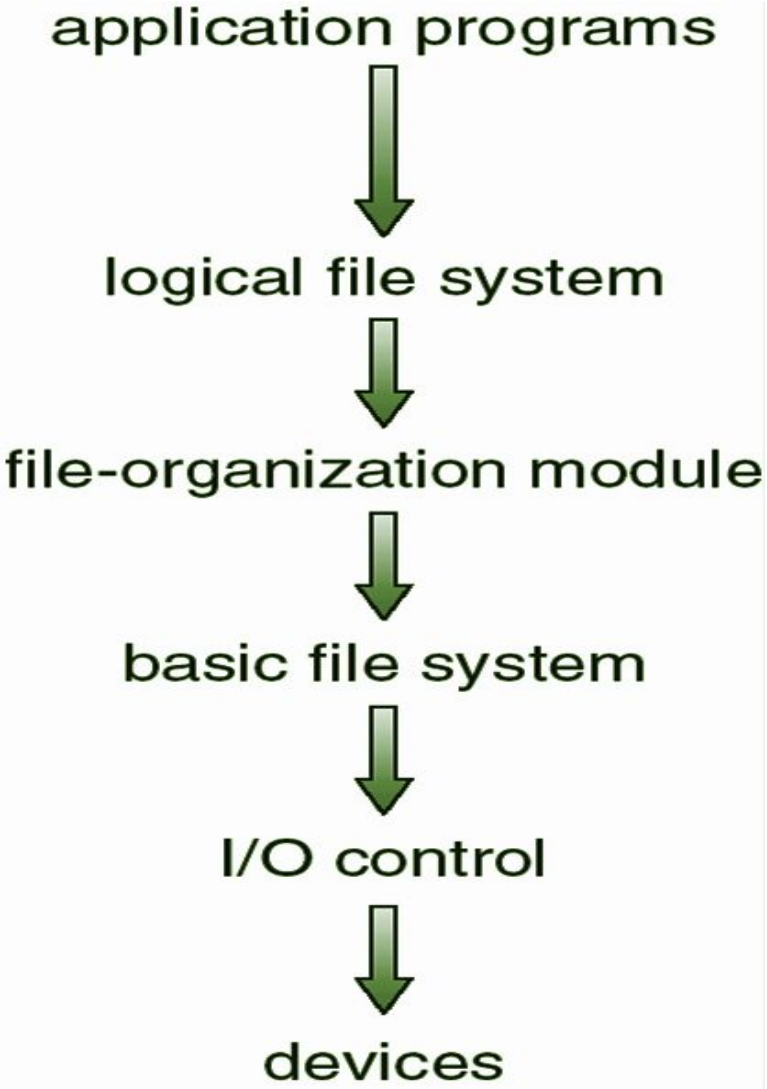
- Disk provides in-place rewrite and random access
    - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

Layered File System



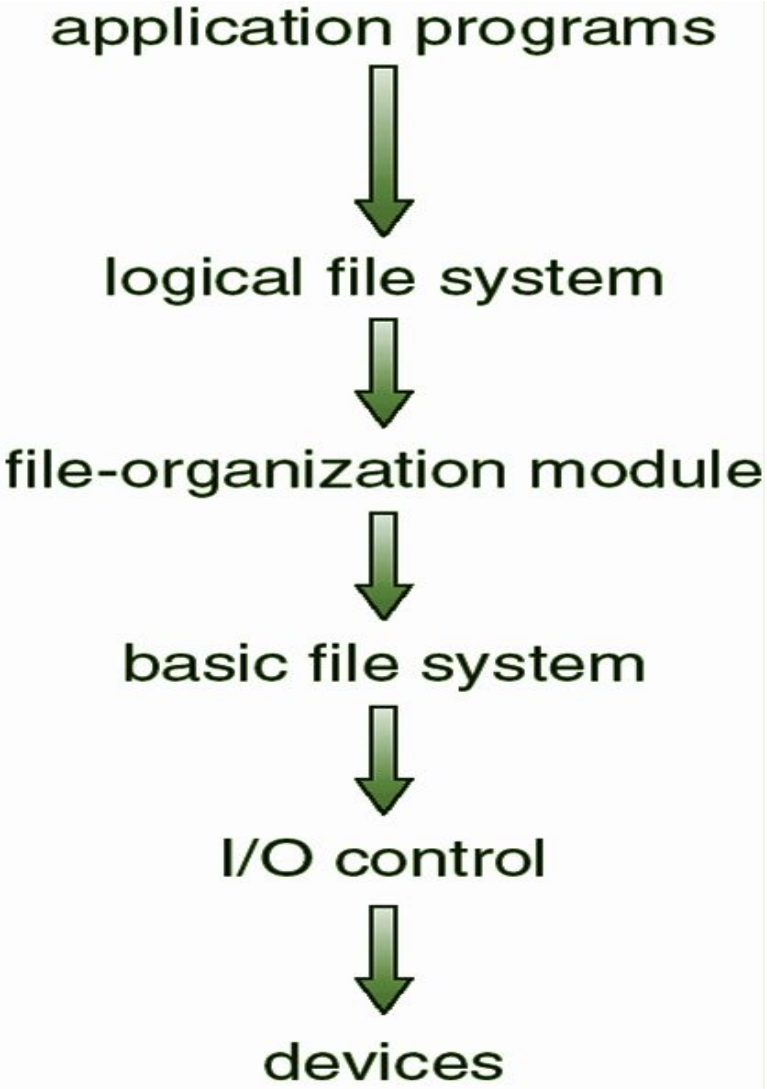
- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like “read drive1, cylinder 72, track 2, sector 10, into memory location 1060” outputs low-level hardware specific commands to hardware controller
- Basic file system given command like “retrieve block 123” translates to device driver

Layered File System



- 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

Layered File System



- 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 **A**utomatic **S**torage **M**anagement - ASM, **F**ilesystem in **U**ser**s**pac**e** - 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

- A storage volume is an identifiable unit of data storage. It can be a removable hard disk, but it does not have to be a unit that can be physically removed from a computer or storage system.

- Each storage volume has a system-unique name or number that allows a user to identify it. In some systems, the physical unit may be divided into several identifiable volumes.

- A partition is a logical division of a hard disk. It is often created to have different operating systems (OSes) on the same hard disk and is generated when a user formats a disk.

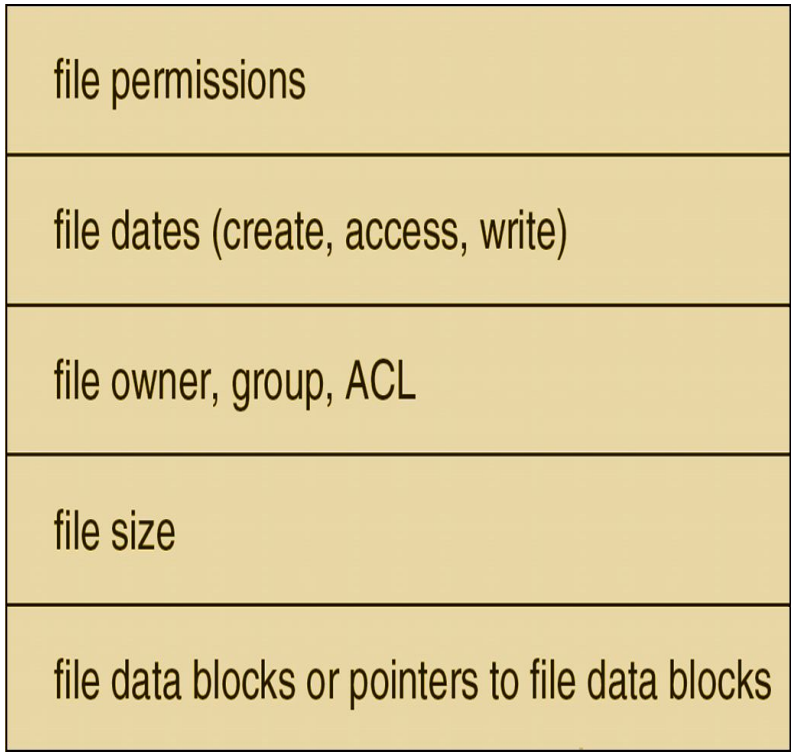
- The main difference between a storage volume and partition is the type of disk used. A volume is created on a dynamic disk -- a logical structure that can span multiple physical disks -- while a partition is created on a basic disk

- A volume is also more flexible than a partition. Storage volumes can expand or contract, and they use mirroring and striping. On a physical server, the OS is typically installed on a partition, while everything else is installed on a volume. In addition, volumes support multiple disks that can be organized into various RAID structures that protect stored data in the event of a hardware failure.

File System Implementation

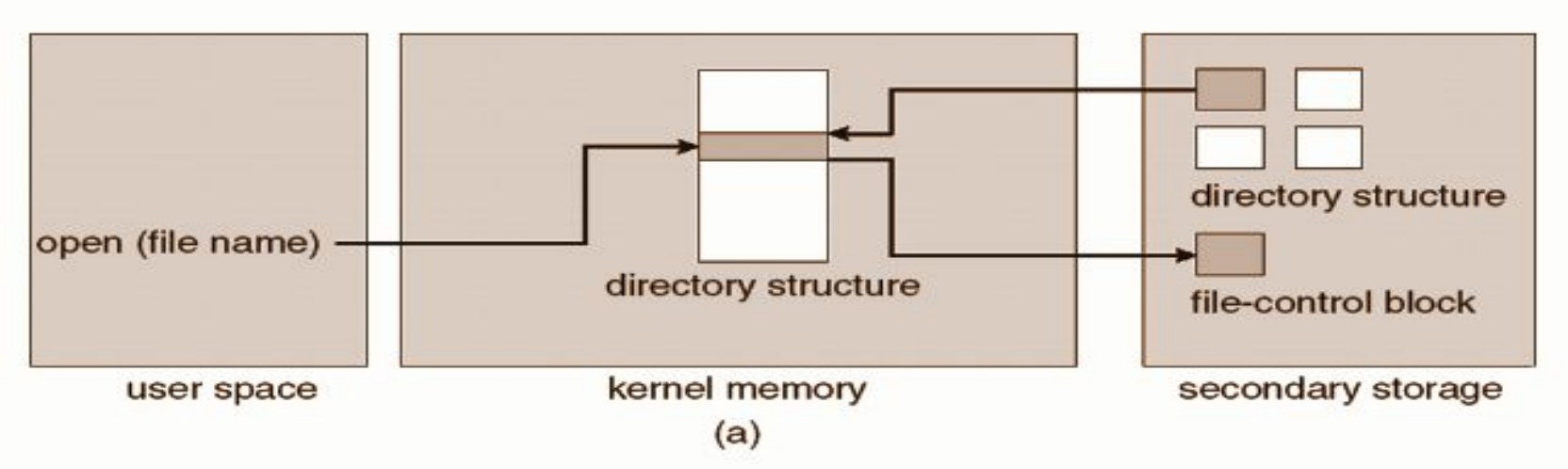
- **Volume** is the label of the drive or the partition.
- **Partition** is a “carved out” segment of the drive memory.
- For example, your computer has a hard drive where Windows / Linux runs and all your programs and probably files are stored. Usually the label (volume) of this drive is “C”.
- A partition is a portion of a hard disk drive that functions as though it is a physically separate drive. After the partition is formatted and assigned a drive letter, the partition is called a drive.
- For example, if you have one hard disk drive and you create two partitions, you can install the operating system and applications on the first partition and store user data on the second partition. If you want to create three partitions, you can use one partition for the operating system, one partition for data folders that are used by applications, and one partition for user data.
- Once a Partition is formatted and given a drive letter is is called a **Volume**. This can be a **physical volume** (on one partition of a physical disk) or a **logical volume** (which sits on RAID Partition for instance that spans multiple physical hard drives).

File System Implementation



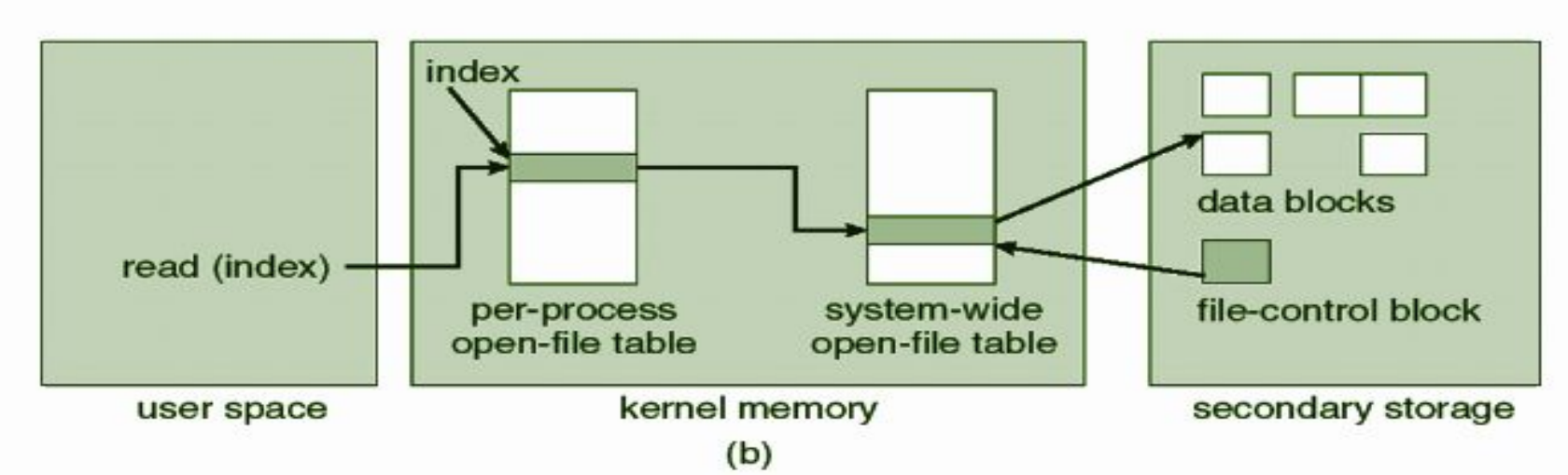
- Per-file **File Control Block (FCB)** contains many details about the file
  - inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures

In-Memory File System Structures



- Mount table storing file system mounts, mount points, file system types
- The above figure illustrates the necessary file system structures provided by the operating systems
- Figure (a) refers to opening 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

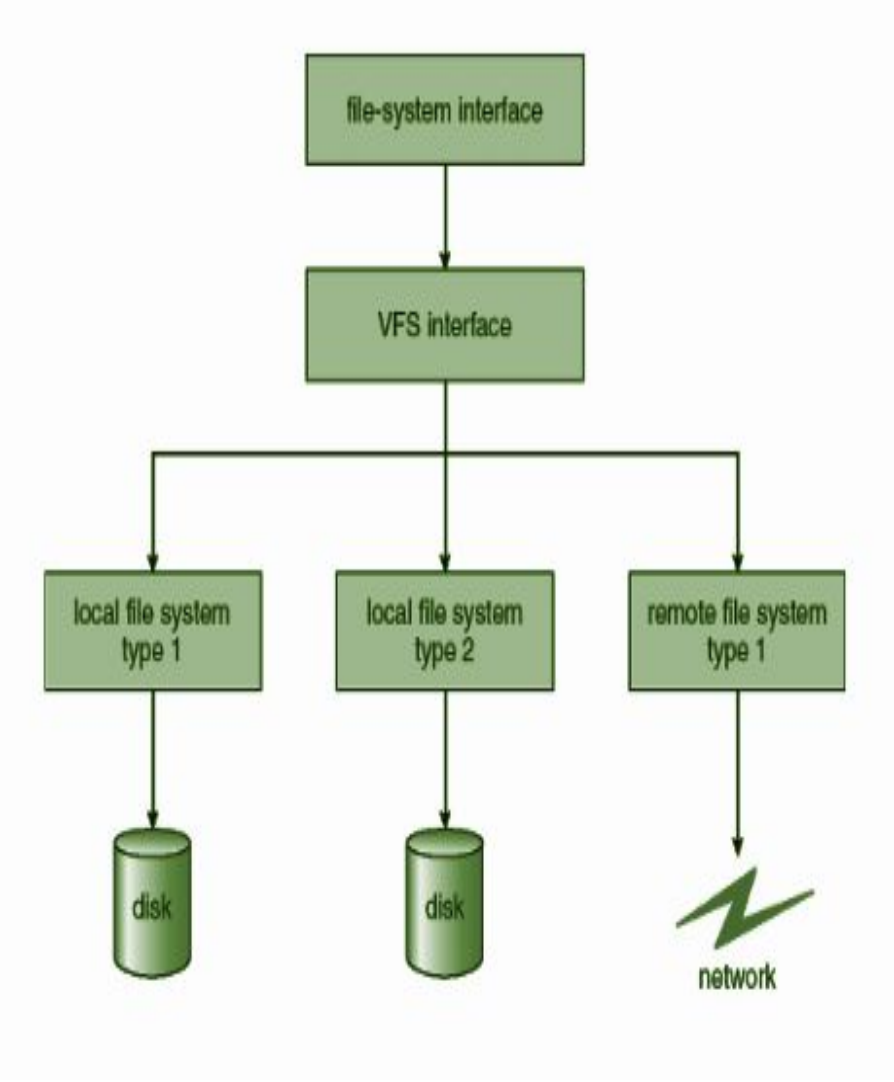


- Mount table storing file system mounts, mount points, file system types
- The above figure illustrates the necessary file system structures provided by the operating systems
- Figure (b) refers to reading a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for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

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

Virtual File Systems 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

Topic Outline

- File-System Structure
- File-System Implementation
- Virtual File Systems
- Virtual File Systems Implementation

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com