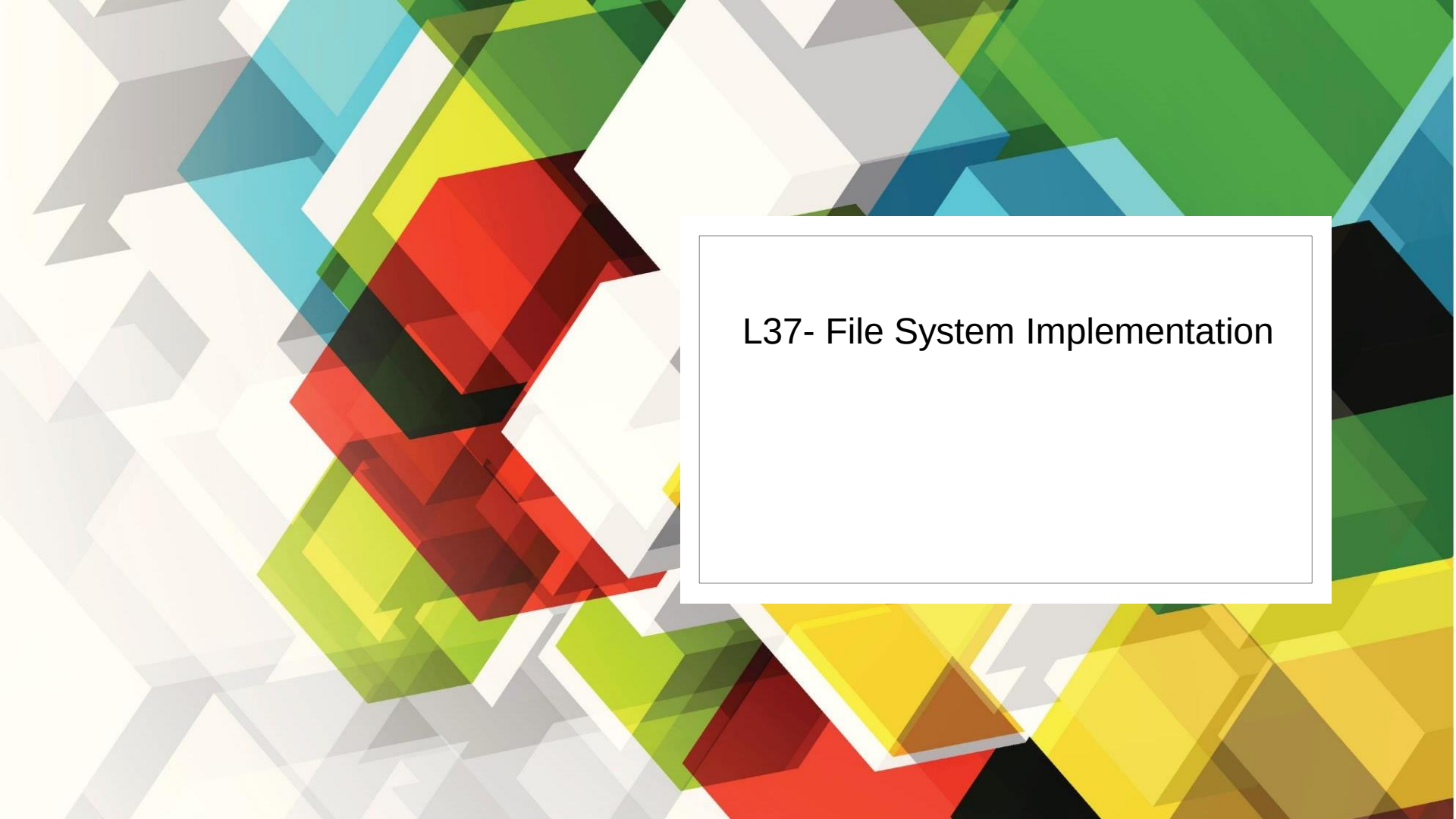

L37- File System Implementation

File-System Implementation

- File-System Mounting
- File Sharing & Protection
- File-System Structure
- File-System Implementation
- Directory Implementation

Objectives

- To explore file-system protection and security features
- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems

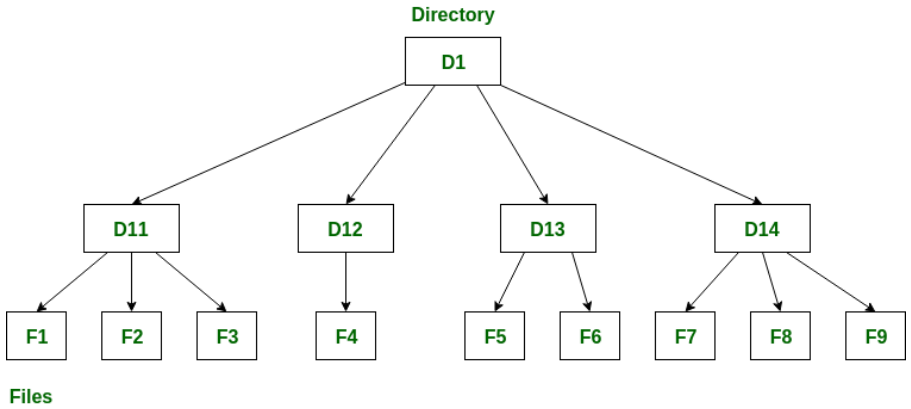
File Concept

- File – Logical Storage Unit
- Types: Data files (numeric, character, binary) & Program files
- Operations: Create, Write, Read, Seek, Delete, Truncate, Open, Close

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Directory Structure

- A collection of nodes containing information about all files
- **Basic operations on directory**
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

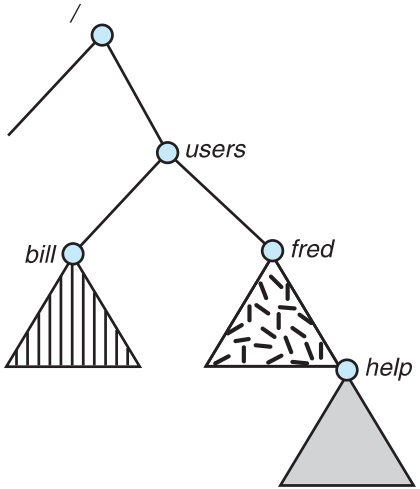


File System Mounting

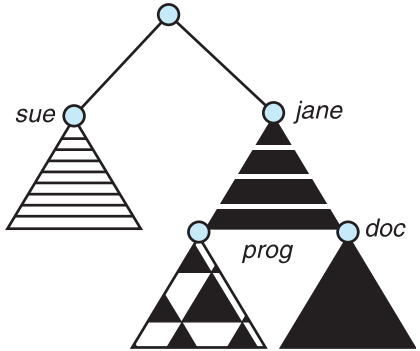
- **Mounting** is a process by which the OS makes files and directories on a storage device available for users to access via the file system.
- OS acquires access to the storage medium; recognize, read and process file system structure and metadata on it.
- The location in file system that the newly-mounted medium was registered is called **mount point**.
- When the mounting process is completed, the user can access files and directories on the medium from the mount point.

File System Mounting

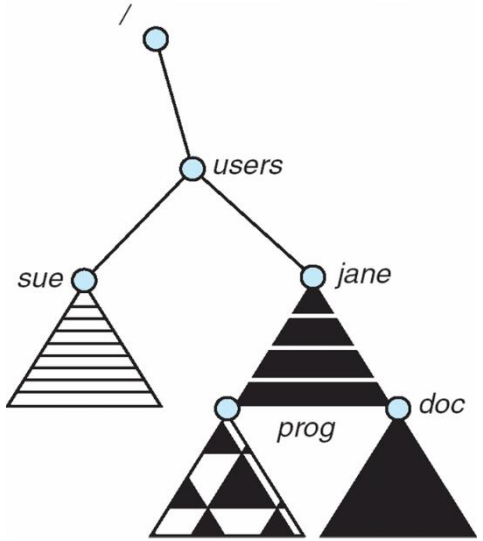
- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**



(a)



(b)



File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
  - **Group IDs** allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers

File Sharing – Remote File Systems

- Standard operating system file calls are translated into remote calls
- **NFS** is standard UNIX client-server file sharing protocol
- **CIFS** is standard Windows protocol
- Distributed Information Systems **(distributed naming services)** such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

- All file systems have failure modes
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols include all information in each request, allowing easy recovery, but less security

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

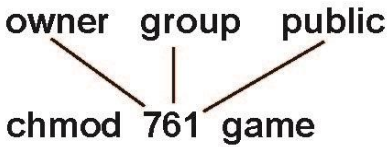
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux [owner, group, public]
- Manager creates a group (G) and add some users to the group.

RWX

a) **owner access** 7 $\Rightarrow$ 1 1 1

b) **group access** 6 $\Rightarrow$ 1 1 0

c) **public access** 1 $\Rightarrow$ 0 0 1

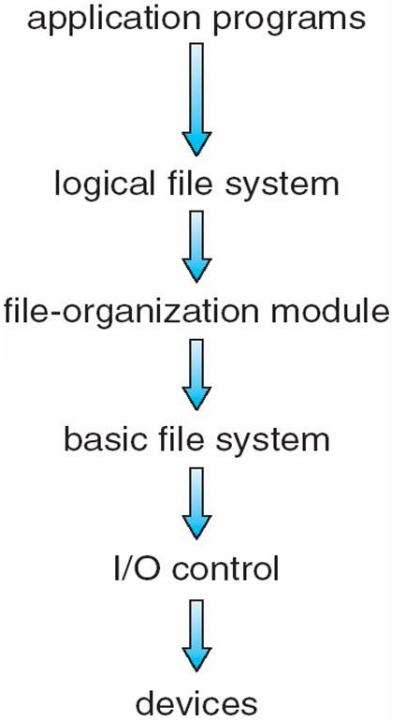


File-System Structure

- File is the Logical storage unit
- **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 physical space for files.
- I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure that has information about a file

Layered File System

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks
  - Directory management & Protection
- **File organization module** understands files, logical address, physical blocks - Translates logical block # to physical block #

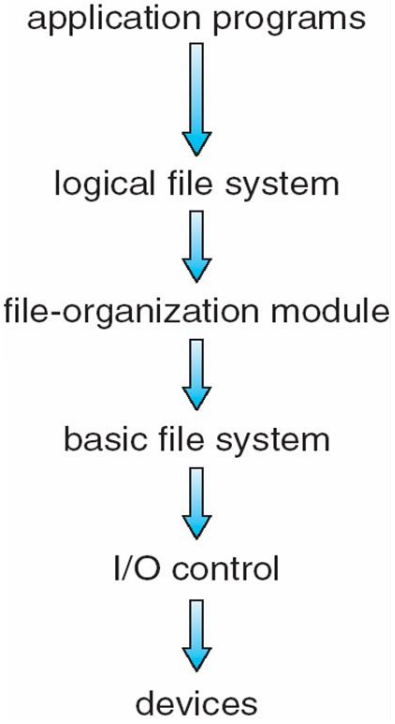


Layered File System

- **Basic file system** issue generic commands to appropriate device driver

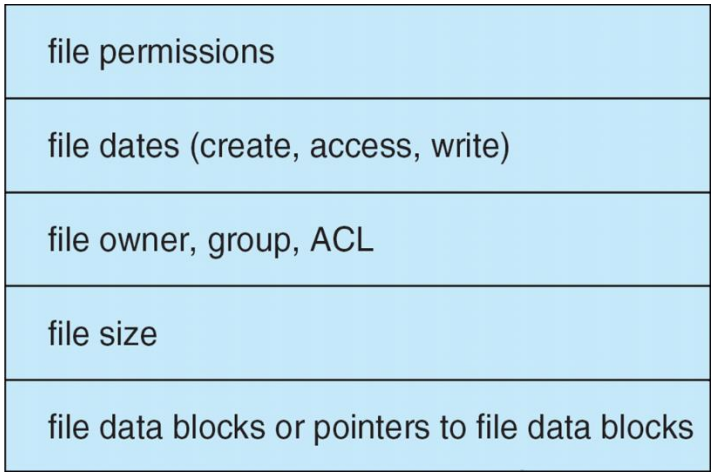
  Eg: *read drive1, cylinder 72, sector 10, into memory location 1060*

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like *read drive1, cylinder 72, sector 10, into memory location 1060* outputs low-level hardware specific commands to hardware controller



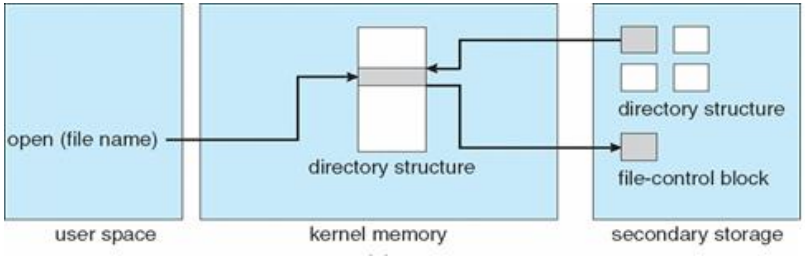
File-System Implementation

- **File Control Block** contains many details about the file
  - inode number, permissions, size, dates

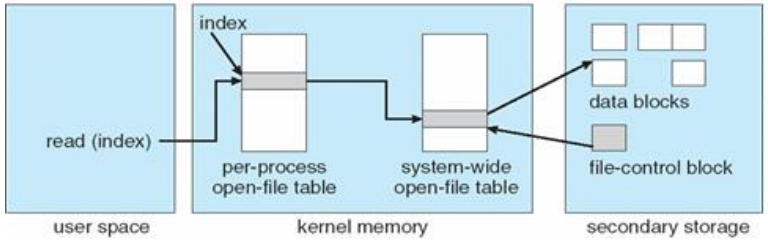


In-Memory File System Structures

- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address



opening a file



reading a file

File-System Implementation

- **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

Partitions and Mounting

- Partition can be a volume containing a file system or 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

Partitions and Mounting

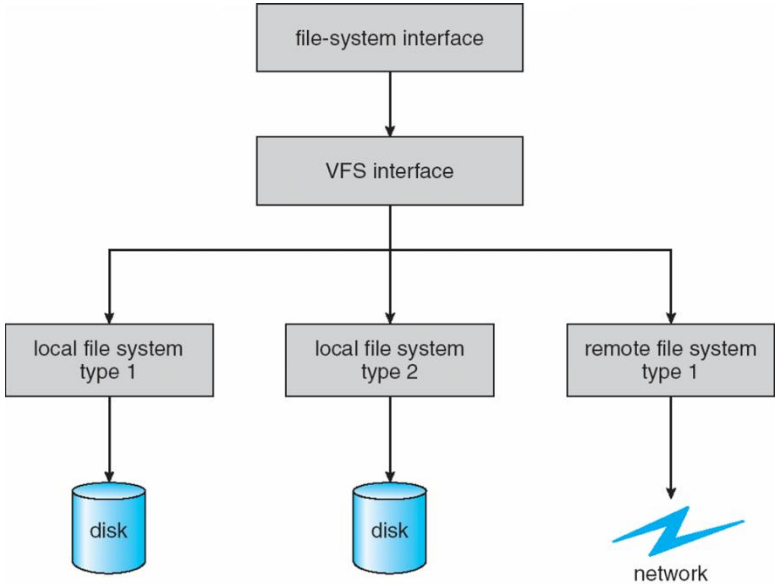
- **Root partition** contains the OS, other partitions can hold other Oses, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
  - Is all metadata correct?
    - If not, fix it, try again
    - 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
  - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems

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



Virtual File Systems

- For example, Linux has four object types:
  - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
  - **`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

Thank You