Operating Systems: File System Interface, Implementation, and System Internals – Part I

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapter 13, 14 and 15)

File Systems

- **File System** - provides efficient and convenient access to disk by allowing data to be stored, located and retrieved easily.
- The file system consists of two distinct parts:
  - A collection of files
  - A directory structure
- Information on disk is stored in the form of files and organized using directories.
- Each file system has its own logic and data structures that reside on secondary storage (disks) to store and organize files.

File System Examples

- File system examples
  - UFS (Unix file system) - UNIX
  - Ext3/Ext4 (extended file system) - LINUX
  - FAT, NTFS etc. – Windows

File

- **File** is the smallest logical storage unit in a file system and is a collection of related information defined by its creator.
  - A file is a sequence of bits, bytes, or a more complex data item.
  - It consists of **logical records** that range from a byte to several bytes.
  - Typically file contents are broken down into physical block sizes (usually 512 bytes or more)
    - File systems I/O performed in group of blocks called **clusters**.

Internal File Structure

- Disks have sectors that determine the physical block size (which is the same or a multiple of the sector size).
- All disk I/O is performed in units of these physical blocks. which are stored on one or more sectors.
- Physical block is not always equal to the logical record of a file.
  - Solution: Packing a number of logical records into physical blocks!

Internal File Structure

- In UNIX all files are streams of bytes.
  - Hence the logical record size is 1 byte.
  - The file system automatically packs and unpacks bytes into physical blocks—say, 512 bytes per block—as necessary.

Internal File Structure

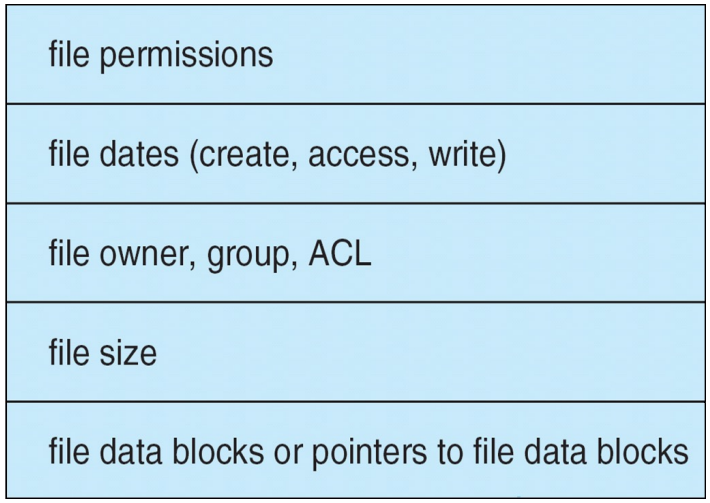
- The packing of logical records can be done either by the user's application program or by the operating system.
- Because disk space is always allocated in blocks, all file systems suffer from internal fragmentation

File

- A file is associated with the following:
- **File attributes** – Name, Identifier (e.g., inode number), text etc.
- **File operations** – Create, write, read, delete, truncate, repositioning within a file.
- **File type** – Executable (.bin, .exe), text file (.txt, .doc)

File Control Block

- Every file has a **file control block (FCB)** that contains all the information about the file (e.g.: ownership, permissions, and location of the file contents.)
- FCB in Unix/Linux systems is called an **inode**.



A typical file-control block

File structure

- Different file types have different file structures.
  - These file structures are used by Operating systems and applications programs to work with them.
  - **All operating systems must support the executable file structure to load and run programs.**
- Supporting different file structure can increase the size of the OS making it hard to manage and work with.
  - Therefore, Unix treats all files as sequence of bytes!

File Access Methods

- File access methods
  - **Sequential Access:** Information in the file is processed in order, one record after the other.
    - Most common
    - Example: editors and compilers are accessed sequentially.
  - **Direct access:** Information in the file is accessed directly (and in no particular order).
    - Example: in databases data is accessed directly.

Directories

- **Directories** – enable segregating files into groups and managing them as groups.
- The directory contains information about the files, including attributes, location, and ownership.
- **Unix treats directories and files as same entities**
- Operations that are to be performed on a directory:
  - Operations regarding a file: Create, delete rename, search.
  - Traverse the file system and list directory contents.
- Various schemes for defining logical structure of a directory exist.
  - Single-level, two-level, tree-structure, Acyclic-Graph Directories and General Graph directories.

Protection

- Files must be kept safe for reliability (against accidental damage), and protection (against deliberate malicious access.)
- Protection is achieved by controlling access to a file.
- A file can be accessed in various ways:
  - **Read, Write, Execute, Append, and Delete**

Access control lists (ACL)

- Is a protection mechanism in which each file/directory has a list of users and the type of access permitted for each user.
- Main issue with ACL is the length.
  - Solution: Condense the list by classifying users.

Access Lists and Groups in Linux

- **Classes of users on Unix / Linux:** Owner, Group, and Others.
- **Mode of access:** R (read), W (write), and X (execute)
- UNIX uses a set of 9 access control bits, in three groups of three
- Each class of user is given RWX permission using 3 bits (bit value 0 = deny access and 1 to grant access).
- File/Directory permissions set using **`chmod`** command.
- Example: Suppose we want to set the below permissions on a file named test.txt:

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	1 1 0
c) **public access**	1	⇒	0 0 1

- The command in UNIX is => **`chmod 761 test.txt`**

Question

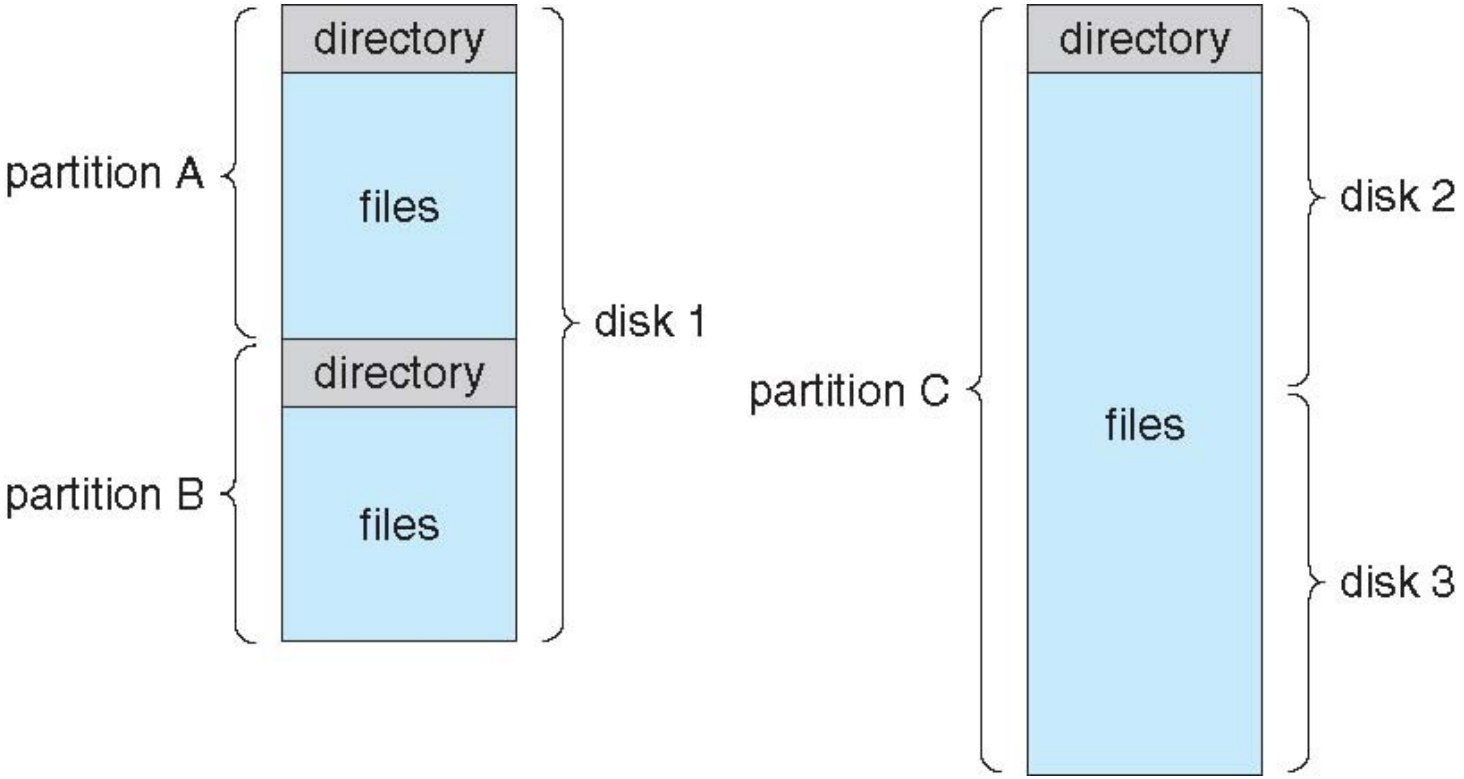
What access rights does the below command specify on the file test.txt?

```
chmod 550 test.txt
```

Disk Structure – Partitions and Volumes

- Disks can be broken up into multiple **partitions, slices, or mini-disks**,
  - Each partition is called a logical disk which can have its own filesystem.
- Multiple physical disks can be combined into one **volume**, i.e., a larger virtual disk
  - This volume can have its own filesystem spanning the physical disks.
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents.**

A Typical File-system Organization



File System Mounting

- Before your computer can use any kind of storage device (such as a hard drive, CD-ROM, or network share), it must be made accessible through the computer's file system.
- This process is called **mounting**.
- An (unmounted) file system is mounted at a **mount point -** the location (typically, an empty directory) within the computer's file system where the file system is to be attached.

File-System Implementation

- Several on-disk and in-memory data structures are used to implement a file system

On Disk Structures

- **Boot control block (boot block in Unix)** is per volume
  - contains information needed by the system to boot OS from that volume. Needed only if volume contains OS and is usually first block of volume.
- **Volume control block (superblock in UNIX)** is per volume
  - Contains volume (or partition) details (e.g.: Total # of blocks, # of free blocks, block size, free block pointers or array etc.)

File-System Implementation

On Disk Structures continued:

- **Directory structure** is per file system
  - Used to organizes the files. In Unix file system, it contains file names and associated inode numbers.
- **File Control Block, FCB,** (per file) containing details about the file.

File-System Implementation - In-Memory File System Structures

In-memory structures are used for both file-system management and performance improvement via caching.

- **Mount table -** contains information about each mounted volume
- **In-memory directory-structure cache** - holds the directory information of recently accessed directories.

File-System Implementation - In-Memory File System Structures

- **System-wide open-file table** contains a copy of the FCB of each open file, as well as tracks the number of processes that have the file open.
- **Per-process open-file table** contains a pointer to the appropriate entry (for the file) in the system-wide open-file table and some other fields.
- **Buffers** hold file-system blocks when they are being read from disk or written to disk.

Creating a new file

- An application program calls the file system to create a file.
- The file system knows the format of the directory structures.
- To create a new file, it allocates a new FCB.
- The system then reads the appropriate directory into memory.
  - Updates it with the new file name and FCB identifier, and writes it back to the disk.

Opening/Reading a file

- The open() call passes a file name to the logical file system.

- The file system first searches the system-wide open-file table to check if the file is already in use by another process.

  - YES => create per-process open-file table entry

  - NO => Search directory structure for the given file name. Once the file is found, its FCB is copied into the system-wide open-file table in memory.

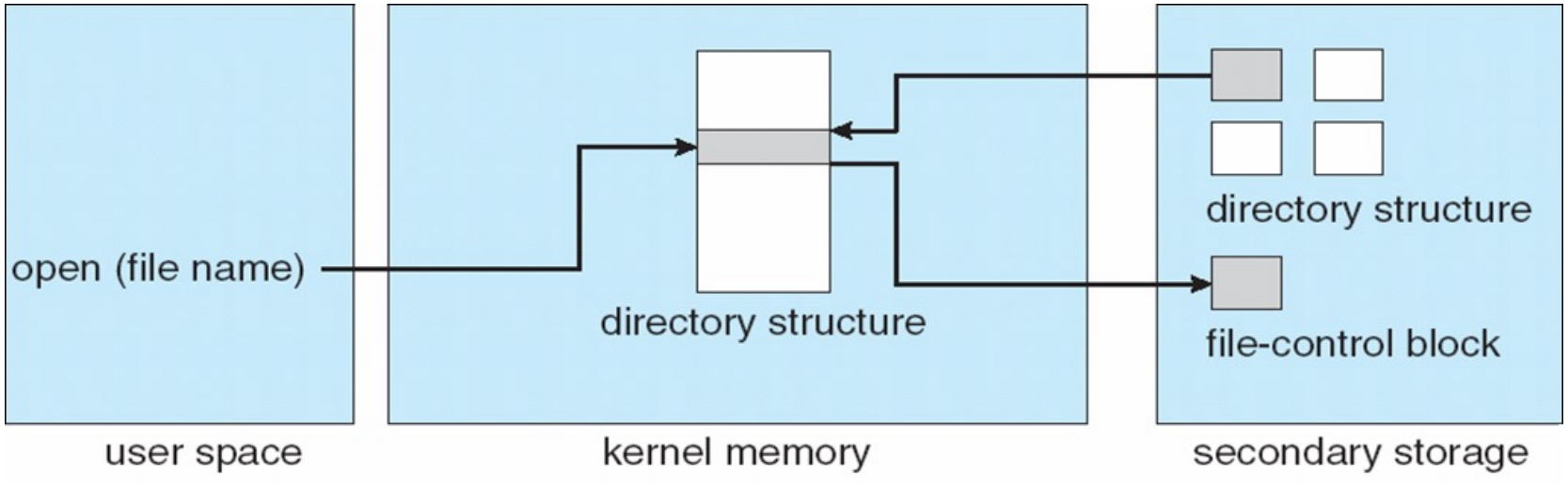
    - System-wide open-file table stores the FCB, and also keeps tracks of the number of processes that have the file open.

Opening/Reading a file

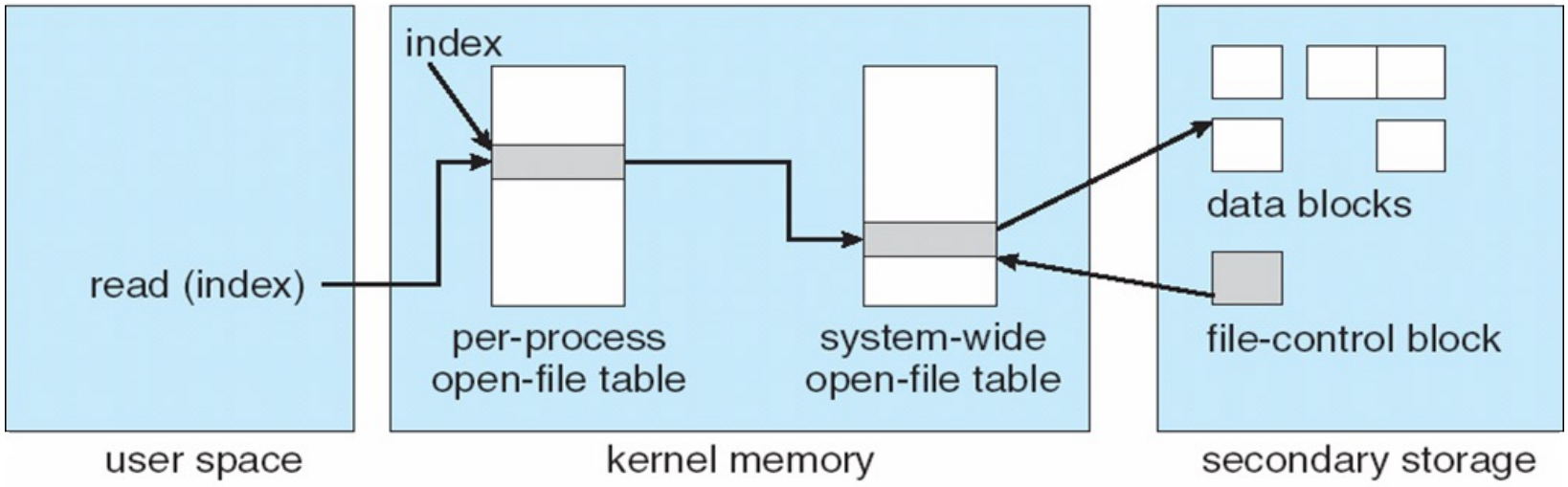
- Update per-process open-file table to store the pointer to the entry in the system-wide open-file table and some other fields (current location, access mode).

- returns a pointer to the appropriate entry (matching the filename) in the per-process file-system table. All file operations use this pointer (UNIX calls it **file descriptor**).

Opening a file – In Memory File System Structures



Reading a file – In Memory File System Structures



Closing a fille

- When a process closes the file, the per-process table entry is removed
- The system-wide entry's open count is decremented.