Lecture-4

File Systems: Organization and Management

Why File Systems are Necessary

- File systems are essential because they ensure data persistence and safety after power loss.
- They abstract away the complexities of disk hardware (like SSD, HDD, or USB).
- They provide a consistent user interface for saving and opening files.
- The file system functions as an "invisible digital librarian," where:
    - Files are the "books"
    - Directories are the "shelves"
    - The OS is the "librarian"

Storage Device Interaction (I/O)

- The OS communicates with storage hardware using **device controllers** and **interrupts**.
- **Interrupts** prevent the CPU from waiting unnecessarily while the disk is working, signaling the OS when an operation is complete.
- File operations heavily rely on I/O between the disk and memory.
- A key method for efficient data movement is **Direct Memory Access (DMA)**, which:
    - Allows data to move directly between the disk and RAM.
    - Bypasses the CPU, freeing it to serve other processes.
- **Asynchronous I/O** further enables the CPU to execute other tasks while data transfers are completing.

File System Structure and Terminology

- The file system provides users with an organized view of storage through **Files** and **Directories**.

Files:

- Defined as a logical unit of storage containing both the actual data and **Metadata** (attributes).

Directories (Folders):

- Containers that organize files and potentially other directories in a **hierarchical tree** structure, starting from a single root directory (e.g., `/` or `C:`).

Metadata:

- "Data about data" — stores extra details such as:
  - File name
  - Size
  - Various timestamps (creation, modification, access)
  - Permissions
  - Ownership details

Inside the Disk: Blocks and Inodes

- The physical storage partition is divided into fixed-size blocks (e.g., 4 KB).
- The file system organizes the partition into key regions:

1. **Superblock:** Contains global file system information.
2. **Free-space map (Bitmaps):** Tracks which blocks are used and which are free.
3. **Inode Table:** Stores the metadata structure for all files.
4. **Data Region:** Stores the actual content of the files.

- The OS organizes files using **inodes** (for metadata) and **data blocks** (for content).
- Each file is associated with one inode, which holds essential metadata (permissions, size, timestamps) and **pointers** that link the inode to the data blocks where the file content resides.

- Example (ext2 file system):
  - Each inode contains **15 disk pointers**.
  - Small files (up to 12 blocks) use **direct pointers**.
  - Larger files use:
    - **Single indirect**
    - **Double indirect**
    - **Triple indirect** pointers, which point to blocks containing lists of data block addresses.

File Access and Path Lookup

- The OS finds a file by resolving its path name (e.g., `/usr/ast/mbox`) step by step.
- The process begins at the root directory (often inode 2).
- The system translates each directory name in the path into its corresponding inode number.
- Each directory is treated as a special file that maps names to inode numbers.
- Once the final inode number is located, the OS uses the pointers within that inode to find the data blocks containing the file's content.

Protection and Journaling

- File systems must protect files from system failure.
- Crashes (like power loss) can leave the file system in an inconsistent state, where data might be saved but metadata hasn't been updated.
- The solution used by modern file systems (e.g., ext3, ext4, NTFS) is **Journaling**.

Journaling Process:

1. Keeps a log (**journal**) of planned changes before applying them to the main file system.
2. Ensures recovery to a consistent state after a crash.

3. Involves:
    - **Write Ahead:** Saving planned changes in the journal first.
    - **Commit:** Applying the logged changes to the file system.
    - **Recovery:** Using the journal upon reboot to complete the changes if a crash occurs.