13.1 File Concept

1. **Introduction to Files**

- Explanation of storing information on various storage media.
- Role of the operating system in providing a uniform logical view of stored information.
- Abstraction of physical properties by defining a logical storage unit: the file.
- Persistence of file contents between system reboots due to nonvolatile storage devices.

2. **Definition of a File**

- Definition: A named collection of related information recorded on secondary storage.
- Importance: Files serve as the smallest allotment of logical secondary storage.
- Representation: Files can represent programs (source and object forms) as well as various types of data.
- Data Types: Numeric, alphabetic, alphanumeric, or binary; free-form or rigidly formatted.

3. **Contents and Structure**

- User Perspective: Files are the method for storing and retrieving data.
- General Purpose: The versatility of files has expanded their usage beyond traditional data storage.
- Types of Information: Files can contain diverse information such as text, programs, photos, music, and videos.
- Defined Structure: The structure of a file depends on its type, with text files organized into lines, source files into functions, and executable files into code sections.

4. **Proc File System**

- Overview: Some operating systems provide a proc file system, utilizing file-system interfaces to offer access to system information.
- Purpose: Facilitates access to system details, including process information, through file-like interfaces.

13.1.1 File Attributes

1. **File Naming**

- A file is named for the convenience of human users, typically represented as a string of characters.
- Naming Conventions: Some systems differentiate between uppercase and lowercase characters in file names, while others do not.
- Independence: Once named, a file becomes independent of the process, user, and system that created it.

2. **Attributes of a File**

- **Name**: The symbolic file name, kept in human-readable form.
- **Identifier**: A unique tag, often a number, identifying the file within the file system; non-human-readable.
- **Type**: Information indicating the type of file, necessary for systems supporting various file types.
- **Location**: Pointer to the device and the file's location on that device.
- **Size**: Current size of the file in bytes, words, or blocks, along with possibly the maximum allowed size.
- **Protection**: Access-control information determining permissions for reading, writing, executing, etc.
- **Timestamps and User Identification**: Data such as creation, last modification, and last use timestamps, along with user identification, useful for protection, security, and usage monitoring.

3. **Extended File Attributes**

   - Some newer file systems support additional attributes like character encoding and security features such as file checksums.
   - Examples: Character encoding, file checksums.

4. **Storage of File Information**

   - Directory Structure: Information about all files is stored in the directory structure on the same device as the files.
   - Directory Entry: Typically consists of the file's name and its unique identifier, which locates other file attributes.
   - Size of Directory: In systems with many files, the directory's size may be significant, potentially in the range of megabytes or gigabytes.
   - Volatility: Directories must match the volatility of files and are usually brought into memory as needed.

13.1.2 File Operations

File operations are essential for interacting with files within an operating system. Let's explore the basic file operations and their implementations:

1. **Creating a File**

   - Allocate space for the file in the file system.
   - Make an entry for the new file in a directory.

2. **Opening a File**

   - All operations, except create and delete, require opening a file.
   - The open call returns a file handle for subsequent operations.

3. **Writing to a File**

   - Specify the open file handle and the data to be written.
   - Update the write pointer to the next write location in the file.

4. **Reading from a File**
    - Specify the file handle and the memory location for reading.
    - Update the read pointer to the next read location in the file.
5. **Repositioning within a File (File Seek)**
    - Reposition the current-file-position pointer to a given value.
6. **Deleting a File**
    - Search the directory for the named file.
    - Release file space and erase or mark the directory entry as free.
7. **Truncating a File**
    - Erase the file's contents while keeping its attributes unchanged.
    - Reset the file length to zero and release its file space.

File Pointer:

- On systems without a file offset as part of read() and write() system calls, the system maintains a current-file-position pointer.
- This pointer keeps track of the last read or write location for each process operating on the file.

2. **File-Open Count:**
    - Tracks the number of opens and closes for a file.
    - Helps manage open-file table entries and prevents running out of space in the table.
    - The file's entry is removed from the open-file table when the open count reaches zero.
3. **Location of the File:**
    - Information necessary to locate the file (e.g., on mass storage, network file server, RAM drive) is kept in memory.
    - Avoids the need to read this information from the directory structure for each operation.
4. **Access Rights:**
    - Each process opens a file with specific access mode (e.g., read-only, read-write).
    - Access rights are stored in the per-process table to allow or deny subsequent I/O requests.

Additionally, some operating systems provide file-locking mechanisms:

- **File Locks:**
    - Prevent other processes from accessing a locked file.
    - Two types: shared locks (multiple processes can access concurrently) and exclusive locks (only one process can access).
    - Operating systems may support mandatory or advisory locking.

- Mandatory locking ensures locking integrity by preventing access until the lock is released.
- Advisory locking relies on software developers to acquire and release locks appropriately.

13.1.3 File Types

1.File Naming and Types:

- File systems often use file extensions to indicate the type of a file.
- The extension, separated by a period, follows the file name.
- Examples include `.docx` for Word documents and `.java` for Java source files.

2. **Extension Functionality:**

- File extensions determine the type of operations allowed on a file.
- Executable files typically have extensions like `.com`, `.exe`, or `.sh` for shell scripts.
- Extensions serve as hints to applications about file contents.

3. **Operating System Specifics:**

- Some operating systems, like macOS, use additional attributes like a file's creator to determine handling.
- macOS automatically opens files with the corresponding application based on creator attributes.

4. **UNIX Approach:**

- UNIX systems may use magic numbers at the start of files to identify types, but this isn't universal.
- File-name-extension hints exist in UNIX but aren't enforced by the OS, primarily aiding users in identification.

13.2 Access Methods:

1. **Files and Accessing Information:**

- Files store information that needs to be accessed and read into computer memory.
- Various methods exist for accessing file data, with some systems supporting multiple access methods.

2. **Sequential Access:**

- Information in the file is processed in order, one record after another.
- Commonly used by editors and compilers.
- Operations include read next() and write next() to read or write portions of the file and advance the file pointer.
- Allows resetting to the beginning and skipping forward or backward by a certain number of records.

3. **Direct Access:**

- File consists of fixed-length logical records.

- Allows rapid reading and writing of records in no particular order.
- Based on a disk model of a file, enabling random access to any file block.
- Operations include read(n) and write(n) to read or write a specific block, where n is the block number.
- Use of relative block numbers simplifies file placement decisions by the operating system.
- Accessing record N in a file translates to an I/O request for L bytes starting at location L * (N), where L is the logical record length.

4. **Support and Limitations:**

- Not all operating systems support both sequential and direct access.
- Some systems require defining a file as sequential or direct upon creation, limiting access methods accordingly.
- Sequential access can be simulated on a direct-access file, but the reverse is inefficient and cumbersome.

13.3 Directory Structure:

1. **Overview:**

- The directory acts as a symbol table translating file names into their control blocks.
- Various operations include search, create, delete, list, rename, and traversing the file system.

2. **Single-Level Directory:**

- Simplest structure where all files are in one directory.
- Easy to support but has limitations with increasing files or users, leading to naming conflicts and difficulty in managing numerous files.

3. **Two-Level Directory:**

- Each user has their directory, avoiding naming conflicts.
- User directories managed by a master directory, allowing each user to have files with the same names.
- Provides isolation but hinders cooperation among users unless explicitly permitted.

4. **Tree-Structured Directories:**

- Generalization of two-level directory to arbitrary height.
- Allows users to create subdirectories, organizing files efficiently.
- Each process has a current directory for easy file access.
- Path names can be absolute or relative, facilitating navigation.

5. **Acyclic-Graph Directories:**

- Enables directories and files to be shared among users.
- Prevents cycles in the directory structure.
- Shared files implemented using links or duplicated directory entries.
- Challenges include managing shared resources and deletion operations.

6. **General Graph Directory:**

- Allows for cycles in the directory structure.
- Algorithms needed to avoid redundant traversals and detect cycles.
- Garbage collection may be necessary to deallocate space when files are deleted.
- Complexity increases with the possibility of cycles, making management more challenging

13.4
Protection Overview:

1. **Reliability:**
   - Ensures data safety from physical damage and system failures.
   - Maintained through duplicate file copies and backup systems.
   - Copies are made regularly to prevent data loss due to accidents or system failures.
2. **Protection:**
   - Safeguards against unauthorized access to files.
   - Implemented through various access control mechanisms.

Types of Access:

1. **Controlled Access:**
   - Limits file access based on the type of operation requested.
   - Operations include read, write, execute, append, delete, list, and attribute change.
   - Higher-level functions may be controlled by lower-level system calls.

Access Control:

1. **Identity-Based Access:**
   - Access dependent on user identity.
   - Implemented through Access Control Lists (ACLs) associating users with specific access rights.
   - ACLs checked when user requests access to a file or directory.
2. **User Classifications:**
   - Owner: Creator of the file with full access rights.
   - Group: Users sharing the file with similar access needs.
   - Other: All remaining users in the system.
3. **Combining Approaches:**
   - Common approach combines ACLs with owner, group, and universe access-control scheme.
   - Provides flexibility in managing access permissions for different user groups.

Protection Challenges and Solutions:

1. **Access List Length:**
   - Long lists may become impractical to manage.

- Condensed version of access-control list introduced to address this issue.

2. **Directory Protection:**
    - Controls file creation, deletion, and listing within directories.
    - Ensures users can only access files and directories with appropriate permissions.

Other Protection Approaches:

1. **Password-Based Access:**
    - Controls file access through passwords.
    - Risk of impracticality with numerous passwords or all-or-none access.
    - Some systems allow password association with subdirectories.
2. **Encryption:**
    - Ensures data security through encryption of partitions or individual files.
    - Effective password management crucial for maintaining security.
3. **Multilevel Directory Structure:**
    - Protects individual files and collections of files in subdirectories.
    - Controls directory operations differently from file operations.
    - Ensures users can access directories and files based on their permissions.
    - Directory listing and file existence detection may be protected operations