

UNIT 7

File System Interface Management

(2 Hrs) — Operating System (CACS251)

7.1 File Concept

Computers store information on several physical media (disks, tapes, flash). The operating system provides a uniform logical view of storage — the **file**. A **file** is a named collection of related information recorded on secondary storage. It is the smallest allotment of logical secondary storage: data cannot be written to storage unless it is inside a file. A file may hold programs (source, object, executable) or data (numeric, text, binary). The part of the OS that deals with files is the **file system**.

7.1.1 File Naming

When a process creates a file it gives it a **name**; after the process terminates the file continues to exist and can be accessed by other processes using that name. Rules vary: MS-DOS (8.3 format: up to eight characters, dot, three-character extension), Windows (up to 255 characters, case-insensitive but case-preserving), UNIX/Linux (up to 255 characters, **case-sensitive**: *file.txt* ≠ *File.txt*). Many systems use a two-part name separated by a period — the **extension** (e.g., *prog.c*, *notes.txt*) which indicates the file type.

7.1.2 File Structure

Files can be structured in several ways:

Structure	Description	Example
Unstructured byte sequence	The OS sees only bytes; the meaning is imposed by user programs. Maximum flexibility.	UNIX, Windows
Record sequence	A file is a sequence of fixed-length records, each with an internal structure; read/write operate on a record	Old mainframes, punch-card era (80-column records)
Tree of records	Records of varying length each with a key field at a fixed position; the tree is sorted on the key for fast lookup	Commercial data-processing, ISAM/B-trees

7.1.3 File Types

Type	Description
Regular files	Contain user information — ASCII (text) files (lines of text, can be displayed/printed) and binary files (executables, archives, images — have an internal structure known to the program).
Directories	System files that maintain the structure of the file system.
Character special files	Related to I/O and used to model serial I/O devices such as terminals, printers, networks.
Block special files	Used to model disks.

Common extension	Meaning
exe, com, bin	Executable program (machine language)
c, java, py, asm	Source code
txt, doc, docx	Textual data, documents
jpg, png, gif, mp3, mp4	Image, audio and video
zip, tar, rar	Archive (compressed)
obj, o	Compiled object code, not yet linked

*Note: An **executable file** in UNIX begins with a magic number that identifies it as an executable, followed by header, text, data, relocation bits and symbol table.*

7.1.4 File Access

- **Sequential access** — bytes or records are read in order from the beginning; cannot skip around, but can be rewound. Used on magnetic tape and convenient when the medium was tape.
- **Random (direct) access** — bytes/records can be read in any order, by position; essential for databases. Two ways of specifying where to start: every read gives the position, or a **seek** operation sets the current position.

7.1.5 File Attributes

Besides the name and data, every file has other information called **attributes** (metadata):

Attribute	Meaning
Name	Symbolic, human-readable name
Identifier	Unique tag (number, e.g., i-node number) identifying the file within the file system

Attribute	Meaning
Type	Needed for systems that support different file types
Location	Pointer to device and to the location of the file on that device
Size	Current size in bytes/blocks and maximum allowed size
Protection	Access-control information: who can read, write, execute
Flags	Read-only, hidden, system, archive, ASCII/binary, temporary, lock flags
Time, date, user identification	Creation time, last-modification time, last-access time and owner — useful for protection, security and usage monitoring

7.1.6 File Operations

Operation	Purpose
Create	Make a new empty file; allocate space and add an entry in the directory
Delete	Release file space and remove the directory entry
Open	Fetch attributes and disk addresses into main memory for fast access
Close	Free internal tables and write back any last block
Read	Read data from the file at the current position
Write	Write data at the current position (overwrites or appends)
Append	Add data at the end of the file
Seek	Reposition the file pointer to a specific place (random access files)
Get / Set attributes	Read or change metadata such as protection mode or time stamps
Rename	Change the name of an existing file
Truncate	Erase contents while keeping the attributes

7.1.7 File Descriptors and the Open-File Tables

When a file is opened, the OS returns a small integer called a **file descriptor** (UNIX) or **file handle** (Windows) which is used in all later operations. The OS maintains:

- A **per-process open-file table** — each entry (indexed by the file descriptor) points to an entry in the system-wide table; holds the current file position pointer.
- A **system-wide open-file table** — holds a copy of the file's attributes (FCB / i-node), **open count** and location on disk. When the open count falls to zero the entry is removed.

Standard descriptors in UNIX: **0** = standard input (stdin), **1** = standard output (stdout), **2** = standard error (stderr).

```
int fd = open("notes.txt", O_RDONLY);    /* returns file descriptor, e.g. 3 */
read(fd, buffer, 100);                  /* read 100 bytes */
lseek(fd, 0, SEEK_SET);                  /* go back to the beginning */
close(fd);                               /* release descriptor */
```

7.2 Directories

To keep track of files, the file system normally has **directories** (folders) which are themselves files. A directory contains entries — one per file — with name, attributes and disk address (or a pointer to an i-node).

7.2.1 Single-Level Directory Systems

The simplest design: **one directory** contains all files for all users (the **root directory**). Easy to implement and fast to look up. *Disadvantages:* **naming problem** — all files must have unique names (with many users a collision is likely); no grouping of related files; hard to protect files of different users. Used in early PCs and simple embedded devices (e.g., digital cameras).

7.2.2 Two-Level Directory Systems

A separate directory for each user (**User File Directory, UFD**) under a **Master File Directory (MFD)**. Solves the name-collision problem between users and offers isolation, but users cannot easily share files or group their own files.

7.2.3 Hierarchical Directory Systems

A generalisation to a **tree of directories**. Users can create as many directories as they need to group files naturally. This is the design used in all modern operating systems (UNIX, Linux, Windows, macOS). The topmost directory is the **root**.

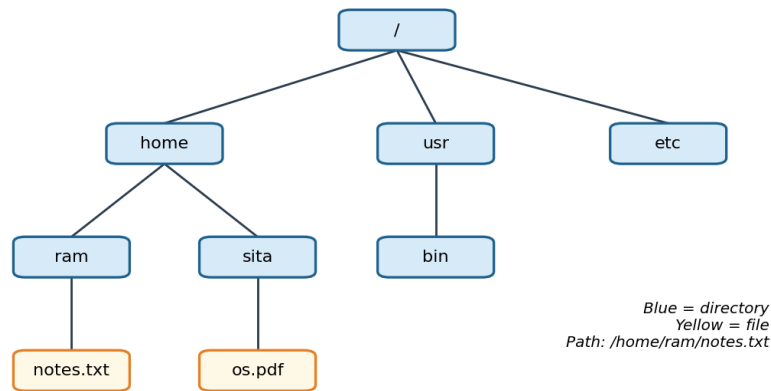


Figure 7.1 — Hierarchical (tree-structured) directory system

Extensions: **acyclic-graph directories** allow sharing of files/directories through **links** (hard link and symbolic link); **general graph directories** allow cycles, which require garbage collection or cycle checks.

7.2.4 Path Names

When the file system is organised as a tree, a way of specifying file names is needed:

Kind	Description	Example
Absolute path name	Path from the root directory to the file; always begins at the root and is unique	UNIX: /home/ram/notes.txt Windows: C:\Users\Ram\notes.txt
Relative path name	Path relative to the working (current) directory	If cwd = /home/ram then <i>notes.txt</i> or <i>../sita/os.pdf</i>

Special entries: **.** (dot) is the current directory and **..** (dot-dot) is the parent directory. The separator is **/** in UNIX and **** in Windows.

7.2.5 Directory Operations

Operation	Description
Create	A directory is created (initially empty apart from . and ..)
Delete	Only an empty directory can normally be deleted
Opendir / Closedir	Open a directory to read its entries; close it to free internal table space
Readdir	Return the next entry in an open directory
Rename	Change the name of a directory
Link	Allow a file to appear in more than one directory
Unlink	Remove a directory entry; the file is deleted when its last link is removed

Operation	Description
Search / List	Find files by name or pattern; list directory contents

7.3 Access Methods

Method	Description	Operations	Used for
Sequential access	Information is processed in order, one record after another; a file pointer moves forward automatically after every read/write	read next, write next, reset, skip	Editors, compilers, tape files
Direct (relative) access	The file is a set of fixed-length logical records; any record can be read/written in any order by its relative block number n	read n, write n, position to n	Databases, airline reservation
Indexed access (extension)	An index in memory maps keys to record positions; the index is searched, then the record is read directly	Search index, then read record	Large indexed data files (ISAM)

Note: Sequential access can be simulated on a direct-access file, but not efficiently the other way round: $cp = cp + 1$ after each read next.

7.4 Protection

The information stored in a computer system must be kept safe from **physical damage** (reliability — backups, RAID) and from **improper access** (protection). Protection mechanisms provide controlled access by limiting the kinds of file access that can be made.

7.4.1 Types of Access

Access type	Meaning
Read	Read from the file
Write	Write or rewrite the file
Execute	Load the file into memory and execute it
Append	Write new information at the end of the file
Delete	Delete the file and free its space
List	List the name and attributes of the file

7.4.2 Access Control List (ACL)

The most general approach: associate with each file (or object) a **list** of (**user/domain, rights**) pairs specifying who may do what. When a user requests access, the OS checks the ACL of the file; if the user is listed with the requested right, access is granted.

Example — Access control lists

File A: (Ram: read, write), (Sita: read), (Hari: none)

File B: (Sita: read, write, execute), (everyone else: read)

- **Advantages:** permission for each user can be very fine-grained; easy to revoke rights for one file.
- **Disadvantages:** the list may be long and variable in length (difficult to store in the directory); constructing the list is tedious; searching for the user takes time on each access.

UNIX condensed ACL: users are classified into **owner**, **group** and **others**, and three bits (**r w x**) are given for each class — nine bits in all. Example: **rwxr-x--x** = 751 in octal (owner: read/write/execute, group: read/execute, others: execute only). The command `chmod 751 file` sets it.

7.4.3 Access Control Matrix

Protection can be viewed as a **matrix** whose **rows are domains** (users/processes) and **columns are objects** (files, devices, printers). Each cell `access(i, j)` lists the rights that domain *i* holds on object *j*.

Domain \ Object	File 1	File 2	File 3	Printer
Ram	read	read, write	—	print
Sita	—	read	read, write, execute	—
Hari	read, write	—	execute	print
Admin	read, write	read, write	read, write, execute	print, manage

- Storing the matrix by **column** (per object) yields an **Access Control List**.
- Storing the matrix by **row** (per domain) yields a **Capability List** — a list of objects and the operations allowed on each, held by the process and protected by the OS.

Access Control List	Capability List
Attached to the object (column of the matrix)	Attached to the subject/domain (row of the matrix)

Access Control List	Capability List
Easy to see who can access an object; easy to revoke	Easy to see everything a user can access; hard to revoke
Used in UNIX and Windows NTFS	Used in some research/secure systems (Hydra, Amoeba)

Key Takeaways

- ✓ A **file** is a named collection of related information on secondary storage, with attributes (name, type, size, protection, times) and operations (create, open, read, write, seek, close...).
- ✓ Files may be byte sequences, record sequences or trees; types include regular, directory and special files.
- ✓ Directory structures: **single-level**, **two-level** and **hierarchical** (tree) — with **absolute** and **relative** path names.
- ✓ Access methods: **sequential**, **direct** (and indexed).
- ✓ Protection uses **types of access** (read, write, execute...), **ACLs** (per object) and the **access-control matrix** (rows = domains, columns = objects); UNIX uses rwx for owner, group, others.

— *End of Unit 7* —