# UNIT–4 : File Management (PYQ Solution)

## PART–A Answers (≈70 Words Each)

---

### Q1. What are the attributes of files? (2019)

**A file has these attributes, stored in its directory entry or inode:**

- **Name: the human-readable name.**
- **Identifier: a unique number used by the file system.**
- **Type: text, executable, source, image and so on.**
- **Location: pointer to the device and blocks holding the file.**
- **Size: current size in bytes or blocks.**
- **Protection: read, write and execute permissions.**
- **Time, date and user: creation, last modification, last access and owner.**

---

### Q2. What is a file system? What are various operations performed on a file? / Explain the term file system in brief. / What are the various file operations? (2019, 2021, Feb 2024, 2024)

**A file system is the part of the OS that stores, organises, names, retrieves and protects files and directories on secondary storage.**

**Basic file operations:**

- **Create: allocate space and add a directory entry.**
- **Open / Close: load the file's attributes into memory / release them.**
- **Read and Write: transfer data at the current file pointer.**
- **Reposition (seek): move the file pointer.**
- **Delete: free the space and remove the entry.**
- **Truncate: erase the contents but keep the attributes.**
- **Append and Rename.**

---

### Q3. What are the various file accessing methods? (2023)

**There are three main access methods:**

- **Sequential access: records are read one after another in order (read next, write next, reset). Used by tapes and text files.**
- **Direct (random) access: any block number $n$ can be read or written directly (read $n$, write $n$). Used by databases.**
- **Indexed access: an index maps keys to block pointers; the index is searched first and then the block is accessed directly (ISAM).**

---

### Q4. Explain various goals of protection. (2021)

**The goals of protection in an operating system are:**

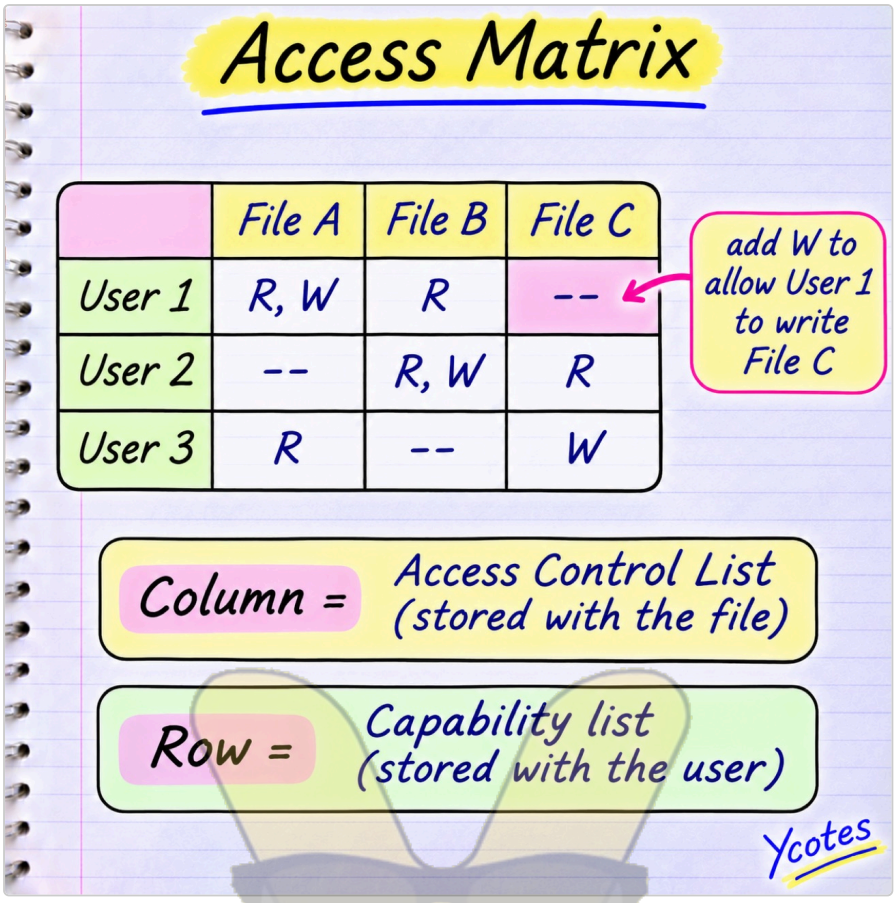
- **Prevent misuse: stop intentional or accidental violation of access restrictions.**
- **Enforce policy: ensure each program uses system resources only in ways allowed by the policy.**
- **Least privilege: give every process only the access it needs.**
- **Detect errors: find hidden errors at the interfaces between subsystems early.**
- **Reliability: increase the reliability of the system with many users.**
- **Separate policy from mechanism: the mechanism says how, the policy says what is allowed.**

---

### Q5. What are access control matrices in file security? (2025)

**An access matrix is a table in which rows are domains (users or processes) and columns are objects (files, devices). Entry $(i, j)$ lists the rights that domain $i$ has on object $j$, such as read (R), write (W), execute (X), delete or copy.**

**It can be stored by columns as an Access Control List (ACL) with each object, or by rows as a capability list with each domain.**


Access Matrix
File A
File B
File C
User 1
R, W
R
--
User 2
--
R, W
R
User 3
R
--
W
add W to allow User 1 to write File C
Column = Access Control List (stored with the file)
Row = Capability list (stored with the user)
Ycotes

# PART–B Answers (≈250 Words Each)

## Q1. Compare the methods by which we can access a file? (2019)

A file can be accessed in three ways. The OS provides one or more according to the file type.

| Basis | Sequential | Direct (random) | Indexed |
|---|---|---|---|
| **Method** | Records read in order from the start | Jump to any block number $n$ | Search an index, then access the block |
| **Operations** | read next, write next, reset | read $n$, write $n$, position to $n$ | search key in index, then read |
| **Speed** | Slow for a record in the middle | Fast for any block | Fast for key search |
| **Structure needed** | None | Fixed-length blocks | Separate index file |
| **Example** | Tape, text file, logs | Database, disk files | Library catalogue, ISAM |
| **Best for** | Batch processing | Large databases, quick lookup | Large files with key-based search |

**Sequential:** a pointer moves forward after each read or write. It is simple, but reaching record 500 means reading the first 499. Rewind is possible.

**Direct:** the file is a numbered sequence of fixed-length blocks, with no order restriction. Sequential access can be built on top of it.

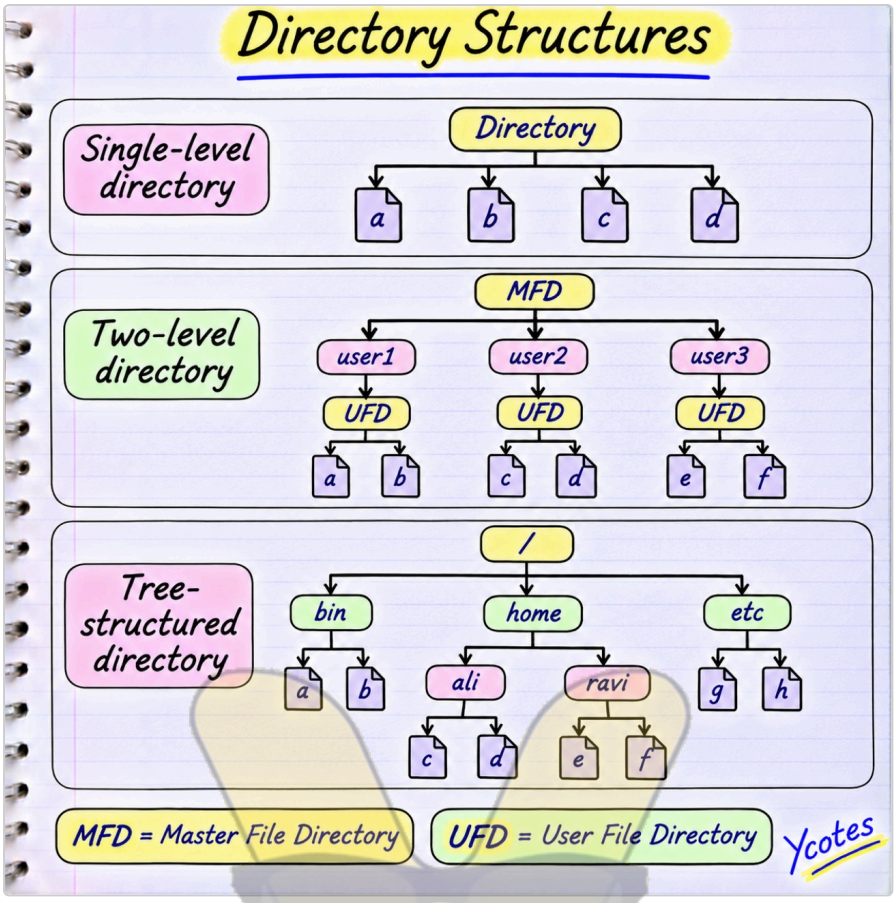
**Indexed:** the small index is kept in memory and holds a pointer for each key. A multilevel index supports very large files.

**Conclusion:** sequential suits streams and logs, direct suits databases, and indexed suits key-based search in huge files.

---

## Q2. Discuss the various directory structures. (2019)

A **directory** is a symbol table that maps file names to file control blocks. The directory operations are search, create, delete, list, rename and traverse.




| Structure | Description | Advantages | Disadvantages |
|---|---|---|---|
| **Single-level** | One directory for all files of all users | Simple, fast search | Name clashes, no grouping |
| **Two-level** | Master File Directory (MFD) points to one User File Directory (UFD) per user | No name clash between users, efficient search | No sub-directories, no sharing |
| **Tree-structured** | Directories inside directories under one root | Grouping, absolute and relative paths, default in most OS | Sharing is awkward |
| **Acyclic graph** | Tree plus shared files and directories through links | Sharing | Dangling links, deletion problems |
| **General graph** | Cycles allowed | Flexible | Needs garbage collection to avoid loops |

**Tree-structured example:** `/home/ravi/os/notes.txt` is the file `notes.txt` in directory `os`, inside `ravi`, inside `home`. A user may create a sub-directory for each subject.

**Conclusion:** modern systems (UNIX, Linux, Windows) use tree-structured directories, with links added for sharing.

---

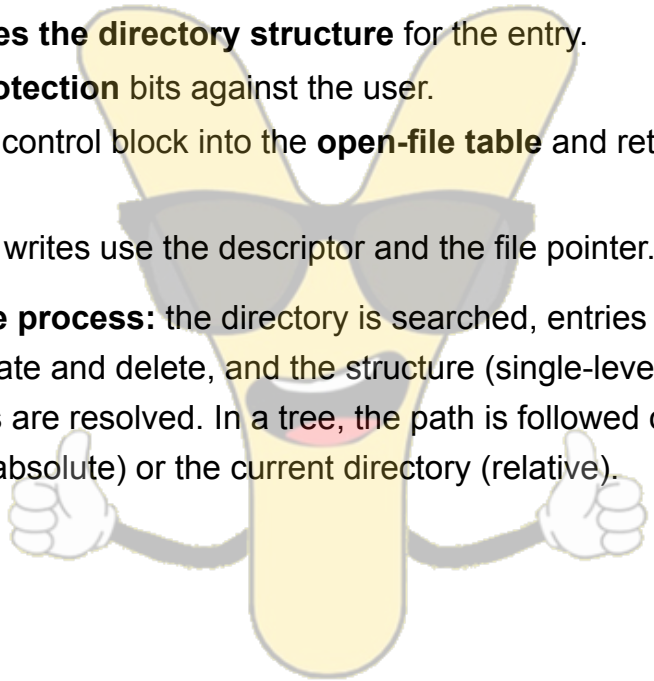
### Q3. Define concept of file operations. Give the process of directory structures and file management. (2021)

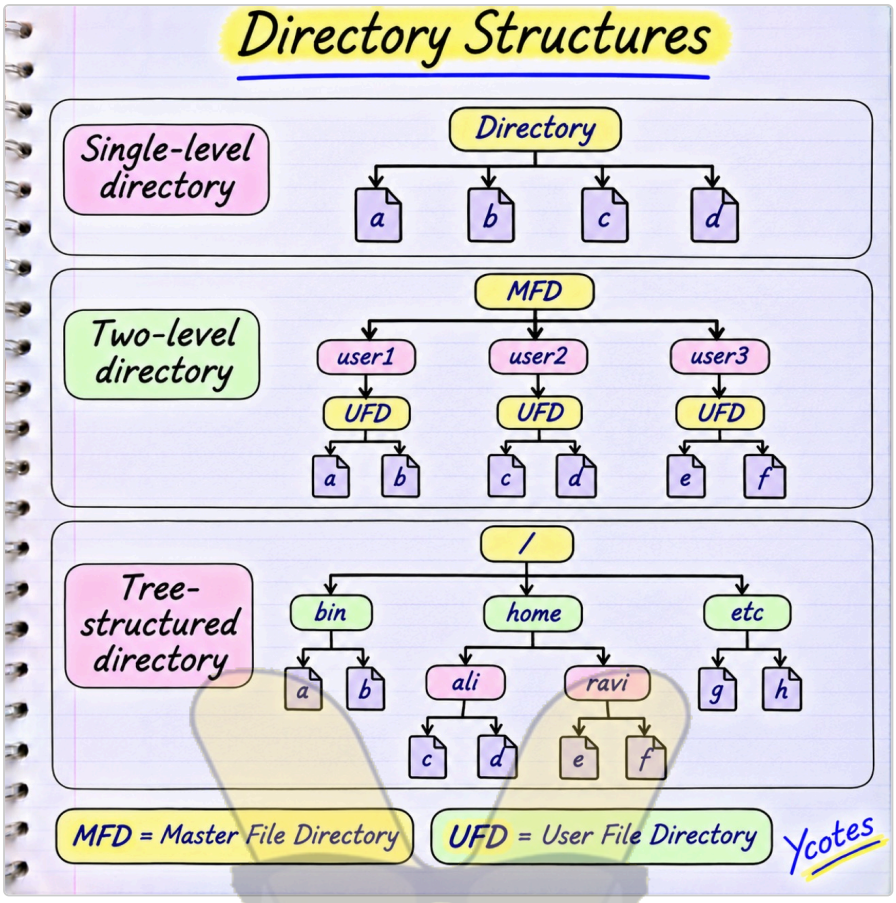
**File operations** are the actions the OS provides on a file through system calls: **create** (find space, add a directory entry), **open** (copy the attributes into the open-file table), **read / write** (transfer at the file pointer), **reposition** (seek), **truncate**, **append**, **rename**, **delete** (free space, remove entry) and **close**.

**Process of a file operation (for example open):**

1. The system call receives the path name.
2. The OS **searches the directory structure** for the entry.
3. It checks the **protection** bits against the user.
4. It copies the file control block into the **open-file table** and returns a **file descriptor**.
5. Later reads and writes use the descriptor and the file pointer.

**Directory structure process:** the directory is searched, entries are added or removed during create and delete, and the structure (single-level, two-level or tree) decides how names are resolved. In a tree, the path is followed one directory at a time from the root (absolute) or the current directory (relative).






**File management** covers allocation of disk blocks (contiguous, linked, indexed), free-space management (bit map, free list), directory handling, protection and recovery.

---

### Q4. What is file management? Explain its types and structures. (2022)

**File management** is the part of the OS that creates, stores, names, retrieves, shares and protects files, and manages the disk space they use. It provides the file abstraction to users.

**Types of files (by extension):**

| Type | Example |
|---|---|
| Executable | .exe, .com |
| Object | .o, .obj |
| Source code | .c, .java |
| Text | .txt, .doc |
| Batch / script | .sh, .bat |
| Archive | .zip, .tar |
| Multimedia | .mp3, .mp4 |

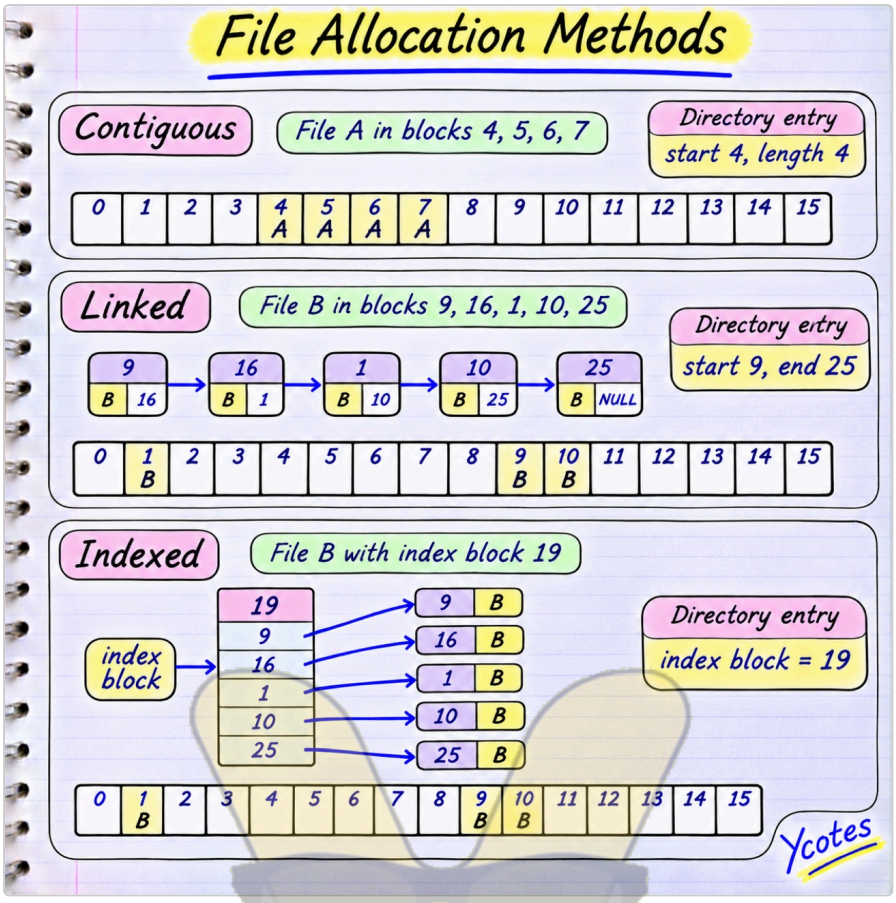
By nature, files are **ordinary files, directory files and special files** (device files in UNIX).

**File structures:**

- **No structure:** a stream of bytes. The OS imposes no meaning; UNIX and Windows use this.
- **Simple record structure:** lines or fixed or variable length records.
- **Complex structure:** formatted documents or relocatable load files.

**Internal structure:** the file is divided into logical records which are packed into fixed-size physical blocks. The last block is often partly empty, which causes **internal fragmentation**.

**Allocation structures:** contiguous, linked and indexed, as in the diagram.




**Functions of file management:** map names to blocks, keep track of free space, control access through permissions, share files between users and recover from failures.

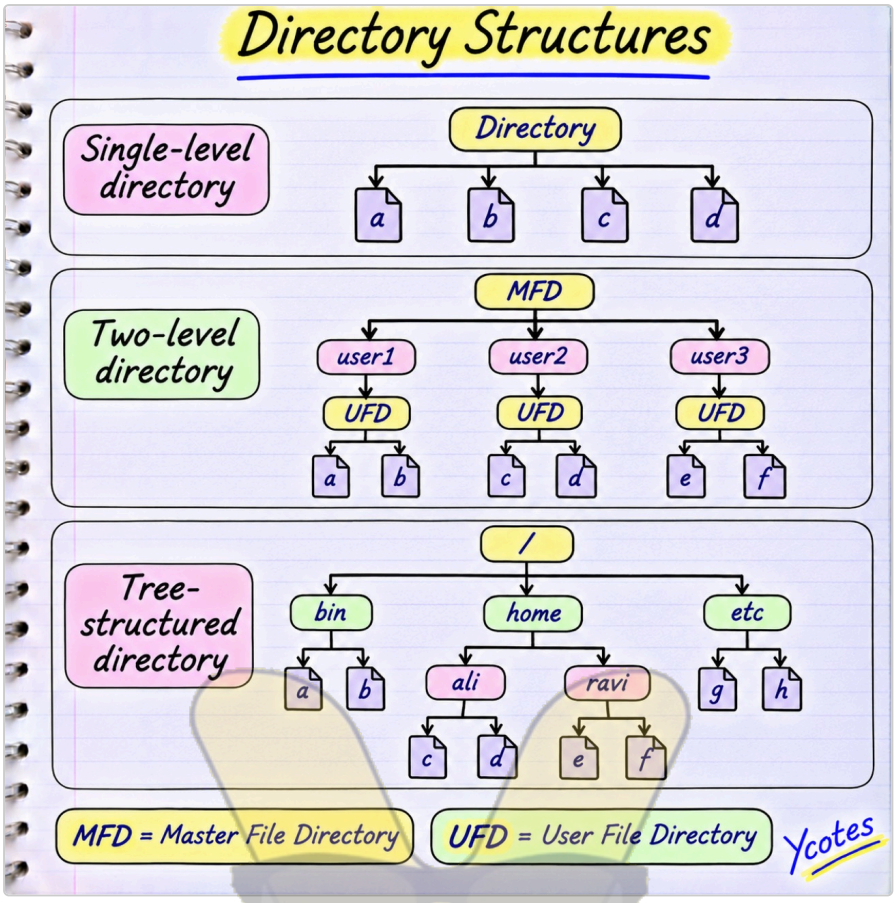
---

### Q5. What is Directory? What are UFD and MFD? Also state the operations that can be performed on a directory. (2024)

A **directory** is a special file that contains information about other files. It acts as a symbol table that maps each file name to its file control block (attributes and disk location).

**MFD (Master File Directory):** the root directory in a two-level scheme. It is indexed by user name and each entry points to that user's UFD.

**UFD (User File Directory):** the private directory of one user, listing only that user's files.




**How they work:** when a user logs in, the system searches the MFD for the user name and then searches only that user's UFD. Two users may therefore use the same file name without conflict. The drawback is that users cannot easily share files or make sub-directories.

**Operations on a directory:**

- **Search** for a file.
- **Create** a file (add an entry).
- **Delete** a file (remove the entry).
- **List** the directory contents.
- **Rename** a file.
- **Traverse** the file system (visit every directory and file).

---

### Q6. If a file system is corrupted due to a crash, what steps would an OS take to recover? Explain. (2025)

A crash in the middle of an update can leave the directory, free-space map and inodes **inconsistent**. For example, a block may be marked used but owned by no file, or two files may own the same block.

**Recovery steps:**

1. **Reboot and run a consistency checker** (`fsck` in UNIX, `chkdsk` in Windows). It scans all inodes, directories and free lists.
2. **Repair inconsistencies:** reclaim lost blocks, correct link counts, resolve duplicate allocations, and move orphan files to the `lost+found` directory.
3. **Journal replay:** journaling file systems (ext4, NTFS) keep a log of intended changes. Committed records are replayed and incomplete ones are discarded, so recovery takes seconds instead of a full scan.
4. **Restore from backup or snapshot** if data or metadata is beyond repair.
5. **Prevention:** careful write ordering, synchronous metadata writes, journaling, RAID and a UPS.

**Conclusion:** `fsck` gives a full consistency repair, journaling gives fast recovery and backups protect against real data loss.

---

### Q7. In a multi-user system, how can file security and user authentication prevent unauthorized access to sensitive logs? (2025)

**Authentication** proves who the user is; **authorisation** decides what that user may do. Together they protect sensitive logs.

**Authentication measures:**

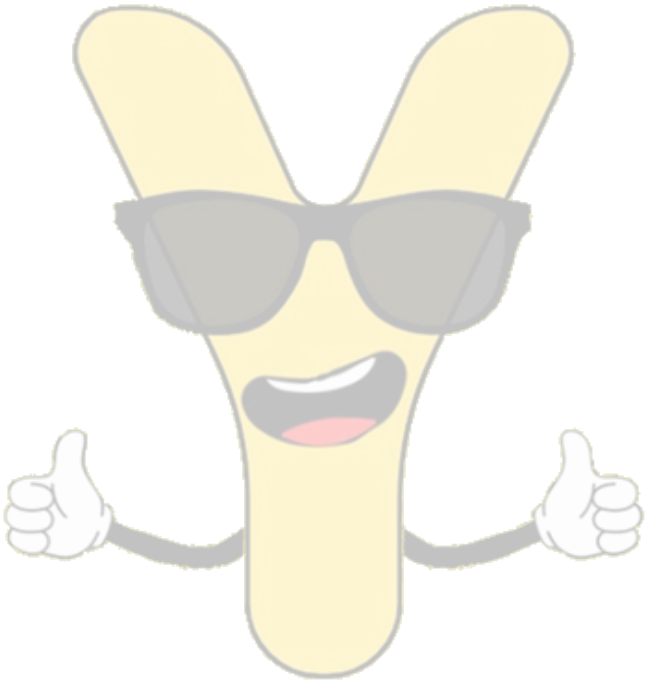
- Passwords stored as salted hashes.
- **Multi-factor authentication** (password plus one-time code or token) for admin accounts.
- Smart cards or biometrics for high-security users.

**File security measures:**

- **Permissions:** UNIX owner, group and others with rwx bits, or Windows ACLs. Keep logs in a directory readable only by the `admin` or `audit` group.
- **Least privilege and role-based access control:** ordinary users cannot even list the log directory.
- **Append-only logs** (`chattr +a`): the logging service may add records but nobody can edit or delete them.
- **Encryption** of logs at rest and in transit.
- **Auditing:** every access attempt is itself recorded.

**Result:** an unauthenticated user cannot log in, and an authenticated but unauthorised user is denied by permissions. A privileged user cannot silently alter evidence because logs are append-only and audited.

---



# PART–C Answers (≈500 Words Each)

## Q1. Explain the followings - (c) File Security. (2022)

**File security** is the set of measures that protect files from unauthorised access, modification, deletion and theft. **Protection** deals with internal access control among users; **security** also covers external threats.

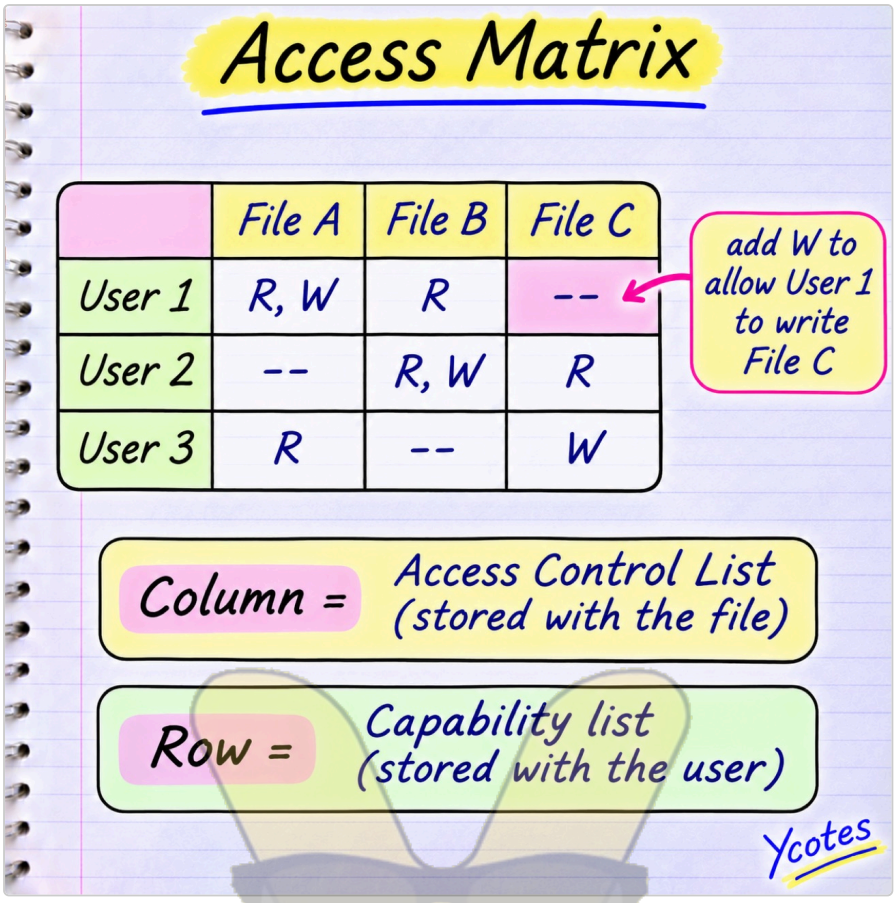
**Security goals (CIA):**

- **Confidentiality:** only authorised users can read data.
- **Integrity:** data is not altered without permission.
- **Availability:** authorised users can always access their data.

**Threats:** unauthorised access, malware and viruses, password guessing, privilege escalation, data theft, accidental deletion and system crash.

**Security mechanisms:**

1. **Authentication:** proving identity by password, one-time password, smart card, biometrics or multi-factor.
2. **Access control (authorisation):** UNIX permissions (rwx for owner, group, others), Windows ACLs and the access matrix.
3. **Encryption:** symmetric or public-key encryption of files and disks.
4. **Auditing and logging:** record who accessed what and when.
5. **Backup and recovery:** regular backups, RAID and snapshots.
6. **Network security:** firewalls and antivirus software.




**UNIX permission example:** `-rwxr-x---` means the owner has read, write and execute, the group has read and execute, and others have none. It is set with `chmod 750 file`.

**Access control list example:** the ACL of a file may say `Ravi: R,W; Admin group: R,W,X; others: none`.

| Mechanism | Protects against |
|---|---|
| Passwords / MFA | Impersonation |
| Permissions / ACL | Unauthorised access |
| Encryption | Theft of data |
| Audit logs | Hidden misuse |
| Backup | Loss of data |

**Advantages:** protects private data, supports multi-user sharing under control and meets legal requirements. **Disadvantages:** adds overhead, can be complex to administer and users may find strong authentication inconvenient.

---

### Q2. Short notes - (iv) Absolute and Relative path name of a file. (2023)

A **path name** specifies the location of a file inside a tree-structured directory system.

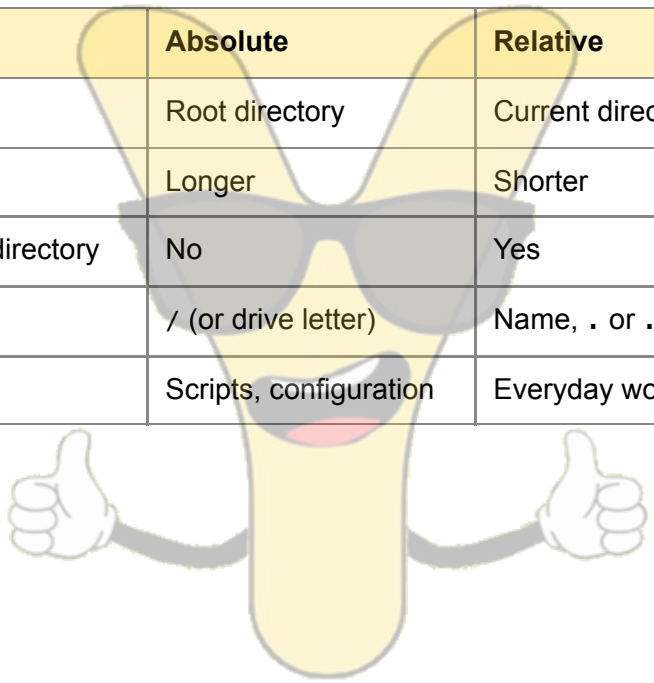
**Absolute path name:** starts from the **root** directory and gives the complete path to the file. It is the same wherever the user is working.

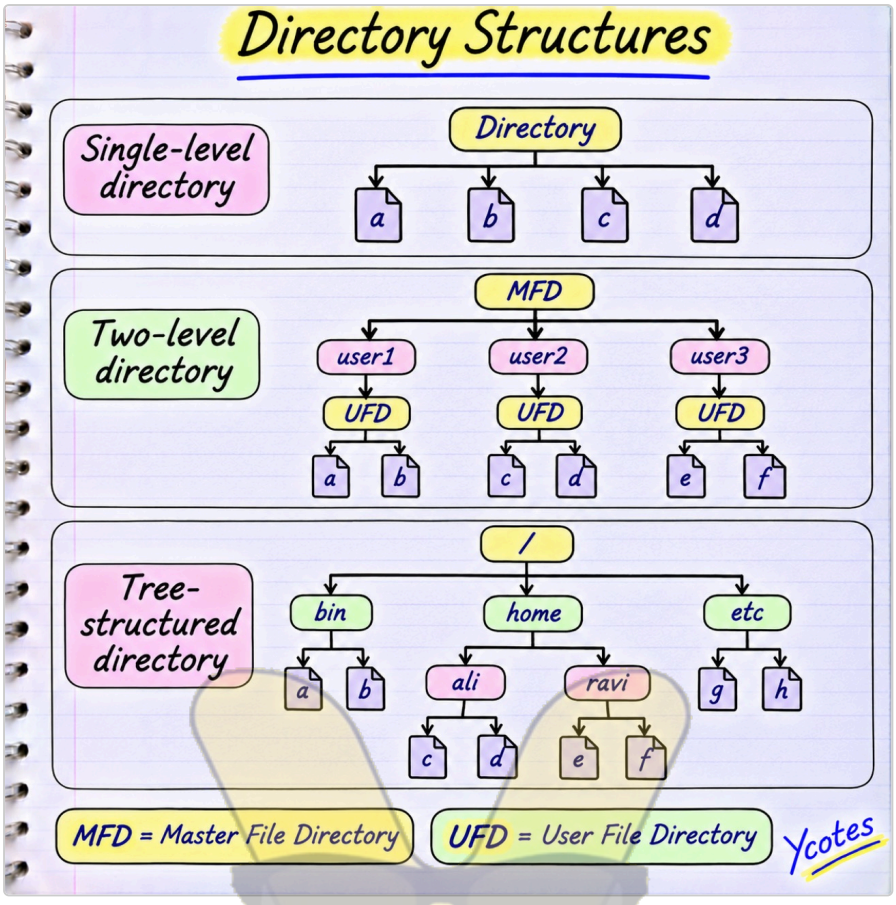
Example: `/home/ravi/os/notes.txt` (Windows: `C:\Users\Ravi\os\notes.txt`).

**Relative path name:** starts from the **current working directory**. It is shorter and changes with the current directory. The symbol `.` means the current directory and `..` means the parent directory.

Example: if the current directory is `/home/ravi`, then `os/notes.txt` refers to the same file, and `../meena/lab.c` refers to `/home/meena/lab.c`.

| Basis | Absolute | Relative |
|---|---|---|
| Starts from | Root directory | Current directory |
| Length | Longer | Shorter |
| Depends on current directory | No | Yes |
| Begins with | `/` (or drive letter) | Name, `.` or `..` |
| Use | Scripts, configuration | Everyday work, portable projects |

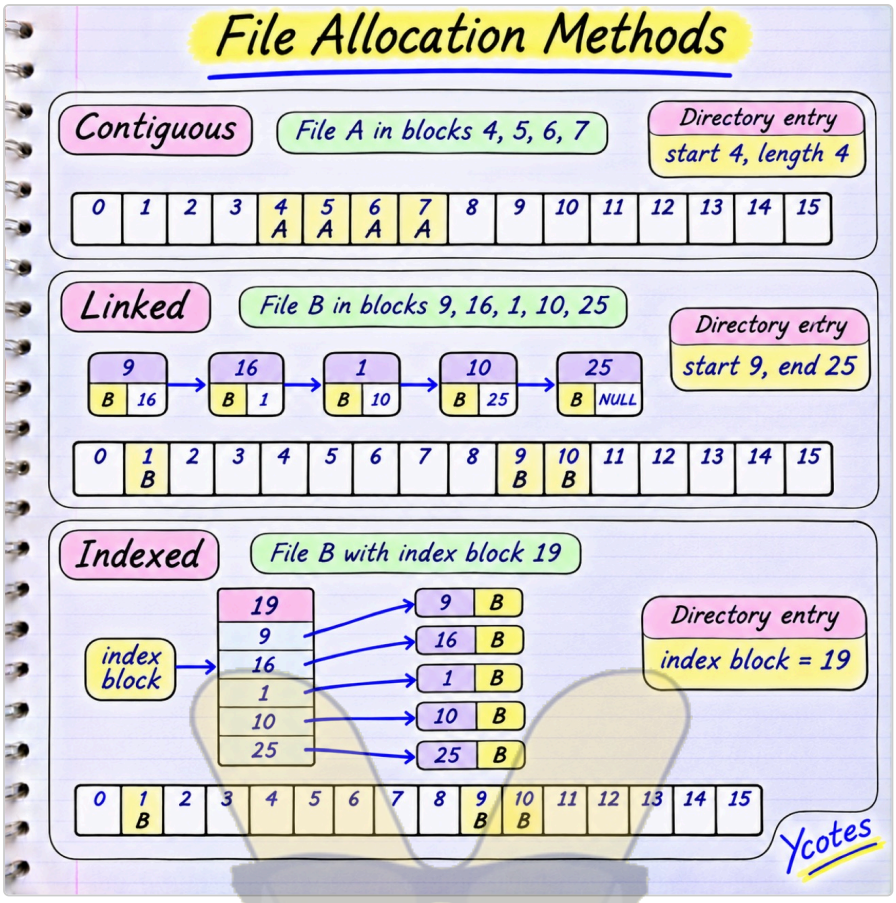





---

**Q3. Discuss the indexed file allocation method with proper example. [4] (2024)**

**Indexed allocation** keeps all the pointers of a file together in one special block called the **index block**. The directory entry stores only the address of the index block. Entry $i$ of the index block points to the $i$th data block of the file.




**Working:**

1. When the file is created, an index block is allocated and all its entries are set to nil.
2. When the $i$th block is written, a free block is found and its address is stored in entry $i$.
3. To read logical block $i$, read the index block, take entry $i$, then read that data block.

**Example:** file `report` has index block **19**, which holds 9, 16, 1, 10, 25.

| Index entry | Block number |
|---|---|
| 0 | 9 |
| 1 | 16 |
| 2 | 1 |
| 3 | 10 |
| 4 | 25 |

The directory stores only 19. To read the third logical block (entry 2), the OS reads block 19, gets block 1, and reads block 1 directly.

**Index size example:** with a 1 KB block and 4-byte pointers, one index block holds $1024/4 = 256$ pointers, so the largest file is $256 \times 1\ \text{KB} = 256$ KB. For bigger files:

- **Linked scheme:** the last pointer leads to another index block.
- **Multilevel index:** an outer index points to inner index blocks, giving $256 \times 256 = 65536$ blocks = 64 MB.
- **Combined scheme (UNIX inode):** 12 direct pointers plus single, double and triple indirect pointers.

**Advantages:**

- Supports **direct access** without searching.
- **No external fragmentation**, because blocks can be anywhere.
- The file can grow easily.

**Disadvantages:**

- The index block is **overhead**, wasteful for very small files.
- Reading a block needs an extra access to the index block unless it is cached.
- Large files need multilevel or linked index blocks.

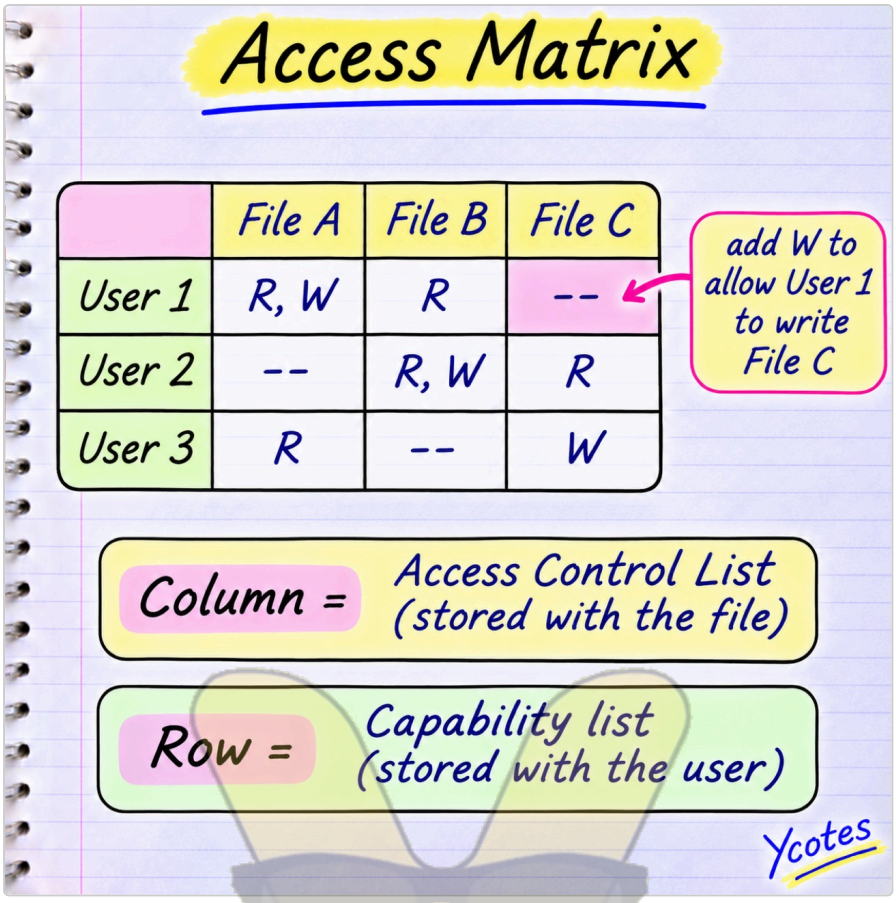
---

### Q4. Given the following access matrix: User/File File A, File B, File C: User 1 R,W / R / --; User 2 -- / R,W / R; User 3 R / -- / W. (a) Can User 2 delete File A? (b) Suggest how access rights can be changed to allow User 1 to write to File C. (2025)

**Access matrix:**

| User / File | File A | File B | File C |
|---|---|---|---|
| User 1 | R, W | R | -- |
| User 2 | -- | R, W | R |
| User 3 | R | -- | W |

Rows are users (domains), columns are files (objects), and –– means no rights.




**(a) Can User 2 delete File A? No.** The entry (User 2, File A) is `--`, so User 2 has no access of any kind to File A. Also, **delete** is not listed as a right for any user in this matrix. Deleting normally needs write permission on the file or its directory, which User 2 does not hold for File A.

**(b) Allow User 1 to write to File C:** change the entry (User 1, File C) from `--` to `W`. This can be done by:

1. The **owner** of File C or an administrator, who edits the entry using `chmod` or an ACL command such as `setfacl -m u:user1:w fileC`.
2. A domain with the **control right** on File C, which may change the rights in that column.
3. User 3, who holds `W` on File C, passing it on using a **copy or transfer right** (written `W*`).
4. Adding User 1 to a **group** that already has write access on File C.

**Updated matrix:**

| User / File | File A | File B | File C |
|---|---|---|---|
| User 1 | R, W | R | **W** |
| User 2 | -- | R, W | R |
| User 3 | R | -- | W |

**ACL of File C (column view):** User 1: W; User 2: R; User 3: W.

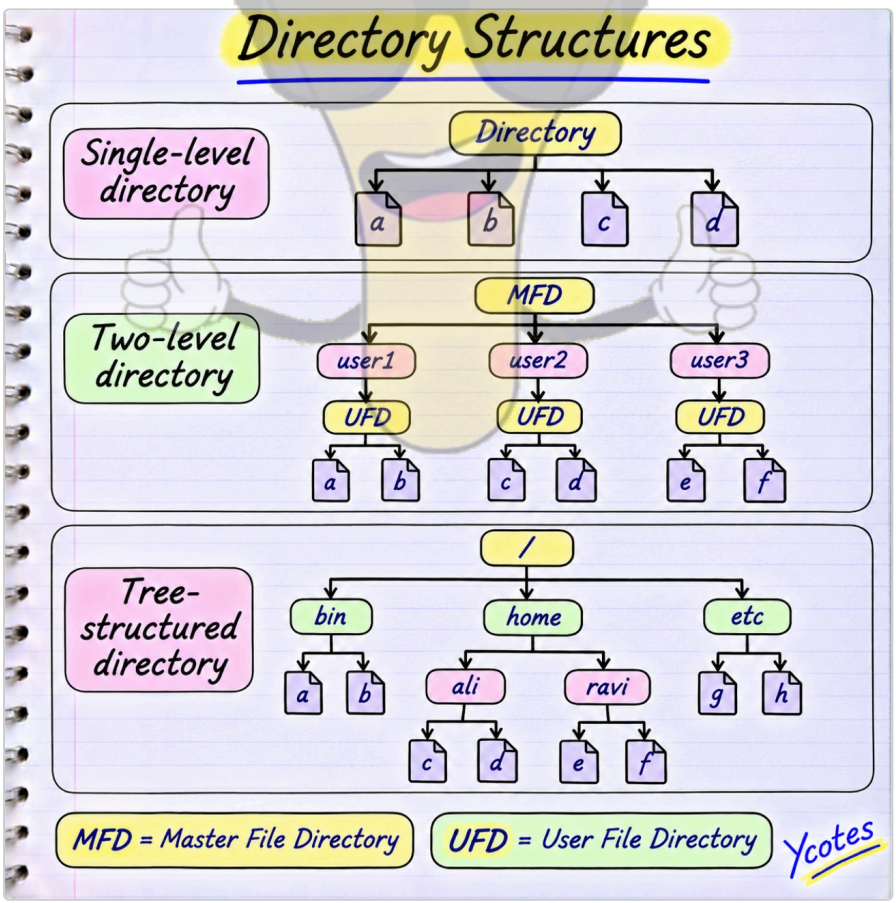
**Capability list of User 1 (row view):** File A: R,W; File B: R; File C: W.

The principle of **least privilege** says grant only W, not R or other rights that User 1 does not need.

---

### Q5. Design a hierarchical directory structure for a university campus server. Include security and access rules. (2025)

A **hierarchical (tree-structured) directory** gives one root with department, role and project folders under it, so files are grouped and path names are clear.




**Proposed design:**

```
/ (root)
 |-- admin/        (registrar, finance, HR)
 |-- faculty/      (one folder per teacher)
 |-- students/     (cs/, ee/, me/ then roll-number folders)
 |-- courses/      (course material, per subject)
 |-- research/     (project groups)
 |-- public/       (notices, forms, timetables)
 `-- backup/, logs/, tmp/
```

**Example paths:**

- Absolute: `/faculty/cs/sharma/notes.pdf`
- Student work: `/students/cs/23CS045/assignment1.docx`
- Relative from `/students/cs`: `23CS045/assignment1.docx`

**Access rules (permission matrix):**

| Directory | Admin | Faculty | Student | Guest |
|---|---|---|---|---|
| `/admin` | rwx | -- | -- | -- |
| `/faculty/<name>` | r (audit) | rwx (owner only) | -- | -- |
| `/students/<roll>` | r | r (own department) | rwx (owner only) | -- |
| `/courses` | rwx | rw | r | -- |
| `/research/<group>` | r | rw (group members) | rw (members) | -- |
| `/public` | rwx | rw | r | r |
| `/logs`, `/backup` | rwx | -- | -- | -- |

**Security rules:**

1. **Authentication:** unique login for every user, strong passwords or multi-factor authentication for faculty and admin.
2. **Role-based groups** (admin, faculty, student, guest) with **least privilege**.
3. **Home directories:** owner has full access, everyone else none.
4. **Quotas** per user to stop disk exhaustion, and `tmp/` cleaned automatically.
5. **Audit logs** of access to `admin/` and `logs/`; logs are append-only.
6. **Encryption** of sensitive folders and **regular backups** to `backup/`.
7. **Sharing** through groups or links, not by opening permissions to everyone.

Brought to you By Ycotes : https://theycotes.com

**Advantages:** easy navigation, clear ownership, simple permissions inherited from the parent directory, and easy backup per sub-tree. **Disadvantage:** deep trees make long path names, and sharing across branches needs links or groups.

---

