Unit 3: Server Functionality & File Management

1. File Storage Systems

1.1 Overview

File storage systems manage how data is stored, organized, and accessed on storage devices. They provide the framework to store files efficiently and securely, supporting operations like reading, writing, and managing file metadata.

1.2 NTFS (New Technology File System)

NTFS is a proprietary file system developed by Microsoft, predominantly used in Windows operating systems.

Core Features:

i. Supports large files and large volumes (up to 16 EB theoretically)
ii. File and folder permissions based on Access Control Lists (ACLs)
iii. File compression and encryption via Encrypting File System (EFS)
iv. Journaling for reliability by logging changes before they are committed
v. Supports symbolic links and hard links
vi. Disk quotas to limit user storage space
vii. Rich metadata: timestamps, attributes, ownership details

Architecture & Internal Structure:

i. Uses a **Master File Table (MFT)** to store metadata about every file and directory
ii. Everything on NTFS is treated as a **file**, including metadata itself
iii. Data is stored in **clusters** (smallest allocatable unit on disk)
iv. Supports **B-tree structures** for fast file lookup

Security Capabilities:

i. Fine-grained permissions using ACLs (read, write, execute, full control)
ii. File-level encryption with EFS
iii. Integration with Windows authentication systems
iv. Supports auditing for tracking file access and modifications

Performance & Reliability:

i. Journaling reduces risk of corruption during crashes or power loss
ii. Self-healing features in modern Windows versions
iii. Efficient handling of large disks compared to older systems like FAT32
iv. Sparse file support for efficient storage of large files with empty regions

Comparison with Other File Systems:

i. **FAT32**:
    - Simpler, but limited (max 4GB file size, no permissions)
ii. **exFAT**:
    - Designed for flash drives, fewer security features
iii. **ext4**:
    - Common in Linux, strong performance but different permission model

Advantages:

i. High security and access control
ii. Reliable due to journaling
iii. Supports very large storage systems
iv. Advanced features suitable for enterprise environments

Disadvantages:

i. Proprietary (limited full support outside Windows)
ii. Slightly higher overhead compared to simpler file systems
iii. Not ideal for small removable drives (where exFAT is preferred)

Real-World Use Cases:

i. Enterprise Windows servers with structured access control
ii. Personal computers running Windows OS
iii. Databases and applications requiring secure and stable storage

Exam Tips (the kind lecturers love):

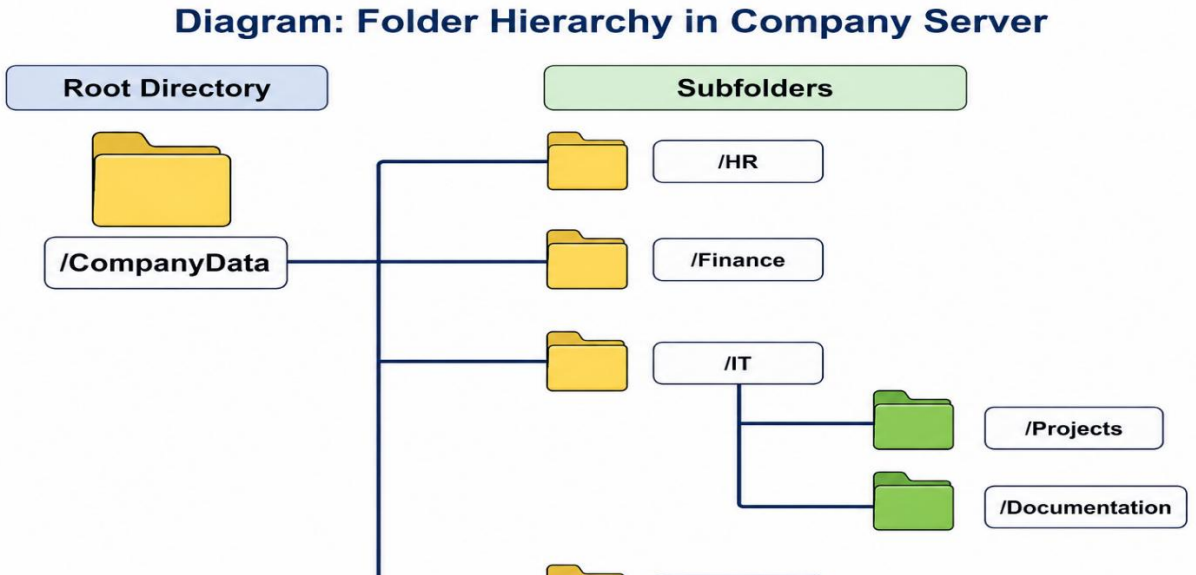
i. NTFS = **Security + Reliability + Large Storage**
ii. Remember **MFT** (very common exam question)
iii. Journaling = **crash protection mechanism**
iv. ACLs = **who can access what**

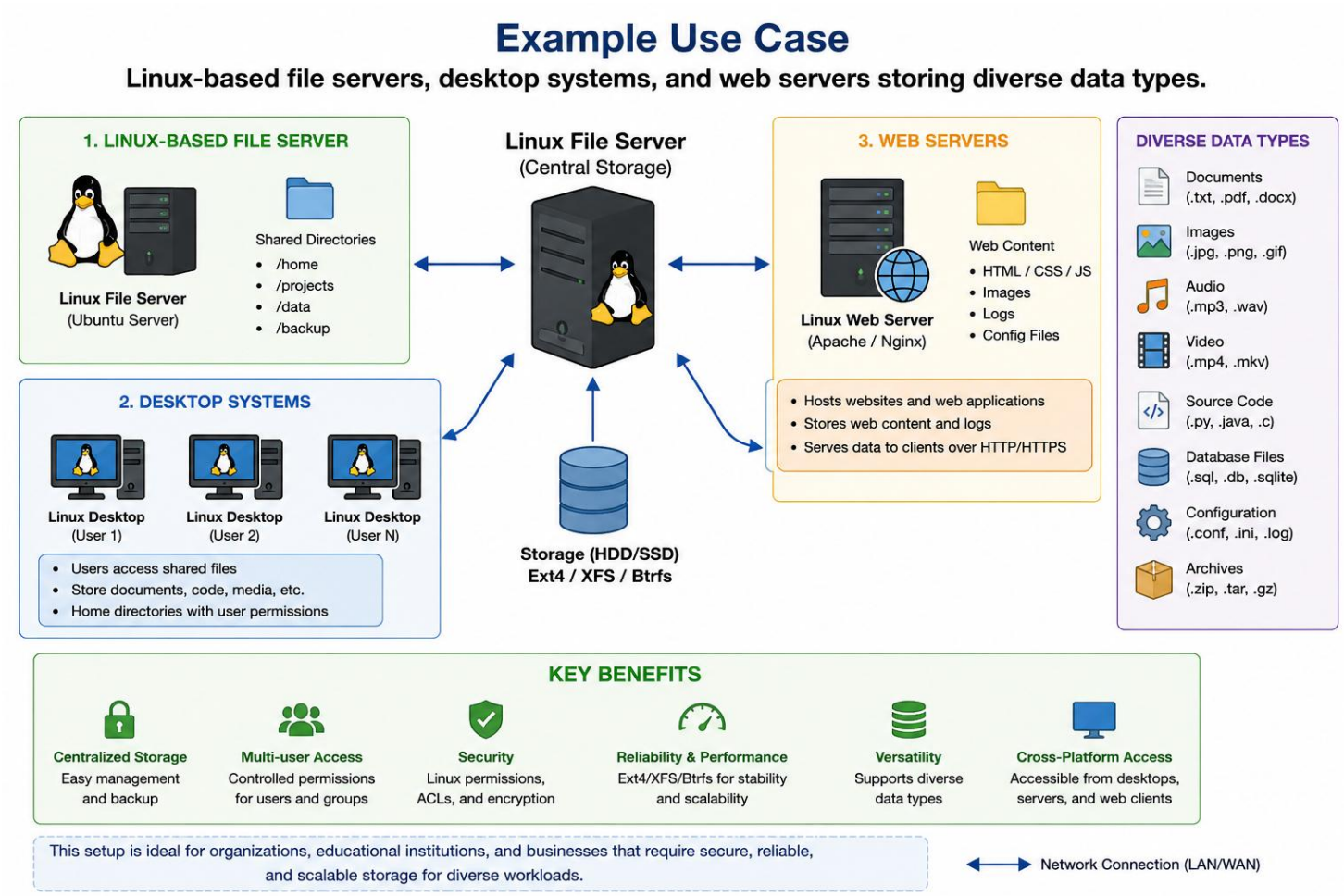
1.3 Linux EXT File Systems

The **EXT** family is the standard Linux file system series evolved through versions:

i. **EXT2** – An early Linux file system without journaling; now largely obsolete.
ii. **EXT3** – Adds journaling for improved fault tolerance.
iii. **EXT4** – Most widely used modern Linux file system offering:
    - Journaling with multiple modes
    - Support for very large files and volumes (up to)
    - Extents (contiguous blocks) for faster performance
    - Backward compatibility with EXT3
    - Delayed allocation to reduce fragmentation

Example Use Case: Linux-based file servers, desktop systems, and web servers storing diverse data types.





1.4 Folder Hierarchy Design

Folder hierarchy refers to the structured arrangement of directories to optimize organization, access, and security.

i. **Design principles:**
   - **Logical grouping:** Arrange folders by department, project, or function
   - **Depth control:** Avoid too deep or too flat hierarchies for ease of navigation
   - **Consistency:** Use consistent naming conventions and folder structures
   - **Security segregation:** Separate folders with different access rights

2. File Retrieval Systems

2.1 Shared Folders

Shared folders allow multiple users or systems to access common data across a network. Key points:

- Typically hosted on servers with directory shares configured
- Accessible via network protocols such as SMB (Windows) or NFS (Linux/Unix)
- Can be public (open access) or restricted via permissions

2.2 Permission-Based Access Control

Enforces controlled access to files/folders based on user identities and group affiliations.

Permission Types:

i. **Read (r)** → View file contents
ii. **Write (w)** → Modify or delete content
iii. **Execute (x)** → Run file (program/script)

Access Control Models:

i. **Discretionary Access Control (DAC):**
Permissions are set by the file owner (flexible but less secure)
Example: NTFS ACLs in Windows
ii. **Mandatory Access Control (MAC):**
System-enforced policies, users cannot override them
Used in high-security environments (military, government systems)
iii. **Role-Based Access Control (RBAC):** *(VERY important for exams)*
Permissions are assigned based on roles (e.g., Admin, Student, Lecturer)
Simplifies management in large systems

Linux File Permissions:

i. Three categories:
   - **User (Owner)**
   - **Group**
   - **Others**
ii. Represented as:
r (read) | w (write) | x (execute)
iii. Example:
drwxr-xr--
   - d → directory
   - rwx → owner (full access)
   - r-x → group (read + execute)
   - r-- → others (read only)

Numeric (Octal) Representation:

i. r = 4, w = 2, x = 1
ii. Add values to form permissions:
   - 7 = rwx
   - 5 = r-x
   - 4 = r--
iii. Example:
755 → Owner full, group read+execute, others read+execute

Key Commands in Linux:

i. chmod → change permissions
ii. chown → change file owner
iii. chgrp → change group ownership
iv. ls -l → view permissions

Advanced Concepts:

i. **Access Control Lists (ACLs):**
Provide more fine-grained permissions than basic Linux model

ii. **Principle of Least Privilege:**
Users should only have access necessary to perform tasks
iii. **Inheritance:**
Subfolders/files can inherit permissions from parent directories
iv. **SUID, SGID, Sticky Bit:** *(exam trap favorites)*
    - **SUID** → run file with owner's privileges
    - **SGID** → inherit group ownership
    - **Sticky Bit** → only owner can delete files (used in shared folders like /tmp)

Security Considerations:

i. Weak permissions can lead to unauthorized access
ii. Overly strict permissions can block legitimate users
iii. Regular audits are essential in enterprise systems

Real-World Example:

University system:

- Lecturers → full access to course materials
- Students → read-only access
- Admin → full control across departments

Exam Tips:

i. rwx + numbers (4,2,1) = **frequent questions**
ii. Know difference: **DAC vs MAC vs RBAC**
iii. Understand permission string interpretation quickly
iv. Remember: **Least Privilege = Security best practice**

2.3 Remote File Access Methods

Enable access to files stored on servers from remote clients over a network.

Core Methods:

Network File System (NFS):
Common in Unix/Linux systems for sharing directories over a LAN

- Works using client-server architecture
- Mounts remote directories as if they are local
- Typically uses port 2049
- Best suited for Linux-to-Linux environments

Server Message Block (SMB):
Native to Windows, also supported on Linux via Samba

- Provides file, printer, and resource sharing
- Uses ports 445 (modern) and 139 (legacy)
- Supports authentication and permissions
- Common in mixed OS environments (Windows + Linux)

SSH File System (SSHFS):
Mounts remote directories securely over SSH

- Uses encryption via Secure Shell
- No need for additional server setup beyond SSH
- Ideal for secure remote administration
- Slower than NFS due to encryption overhead

WebDAV (Web Distributed Authoring and Versioning):
HTTP-based protocol for managing files remotely

- Works over standard web ports (80/443)
- Allows collaborative editing of files
- Often used in cloud and web-based systems

Supporting Concepts:

i. **Mounting:**
Process of making a remote file system accessible locally
ii. **Client-Server Model:**
Server hosts files, clients request access
iii. **Authentication:**
Ensures only authorized users access resources (passwords, keys, tokens)

Comparison Snapshot:

Method	Best For	Security	Speed	Platform
NFS	Linux networks	Low (by default)	High	Linux/Unix
SMB	Windows networks	Moderate	Moderate	Cross-platform
SSHFS	Secure remote access	High	Lower	Linux/Unix
WebDAV	Web/cloud access	Moderate (HTTPS)	Moderate	Cross-platform

Advantages:

i. Enables centralized file storage
ii. Supports collaboration across networks
iii. Reduces data duplication
iv. Allows remote work and system administration

Disadvantages:

i. Network dependency (latency affects performance)
ii. Security risks if misconfigured
iii. Compatibility issues between systems

Real-World Use Cases:

i. University labs accessing shared resources
ii. Company file servers for employees
iii. Cloud storage systems using WebDAV
iv. Remote server management using SSHFS

Security Considerations:

i. Use encryption (SSH, HTTPS) whenever possible
ii. Avoid exposing NFS directly over the internet
iii. Implement firewalls and access controls
iv. Regularly update and patch systems

Exam Tips:

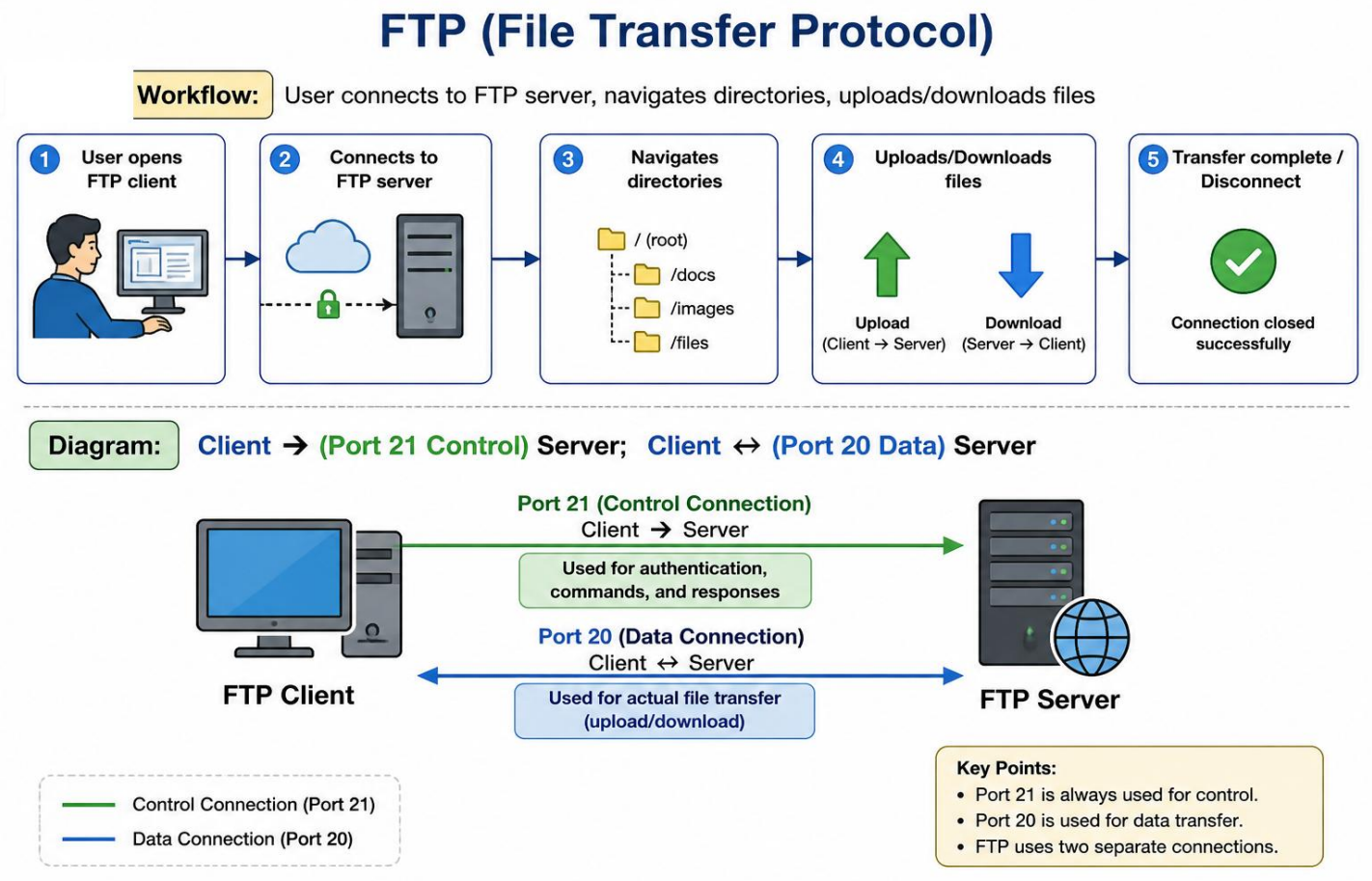
i. NFS = **Linux sharing**
ii. SMB = **Windows sharing**
iii. SSHFS = **secure mounting via SSH**
iv. WebDAV = **HTTP-based file access**
v. Know ports: **NFS (2049), SMB (445), HTTP (80), HTTPS (443)**

3. File Transfer Mechanisms

3.1 FTP (File Transfer Protocol)

FTP is a traditional protocol for transferring files between client and server over TCP/IP networks.

- **Key Characteristics:**
  - Uses separate control (port 21) and data connections (port 20)
  - Supports anonymous and authenticated access
  - Does not encrypt data unless used over VPN or secured connections
- **Example Workflow:** User connects to FTP server, navigates directories, uploads/downloads files



3.2 SMB (Server Message Block)
SMB is a network file sharing protocol widely used in Windows environments for both file and printer sharing.

i. Provides shared access to files, printers, and serial ports
ii. Integrated with Windows authentication and permissions
iii. Accessible through mapped network drives
iv. Samba enables SMB services on Linux/Unix
v. **Example:** Mapping shared folder \\ServerName\SharedFolder on Windows

3.3 Secure Transfer Methods
To enhance security in file transfers, encryption and secure protocols are employed.

SFTP (SSH File Transfer Protocol):

i. Runs over Secure Shell ensuring **encrypted authentication and data transfer**
ii. Supports **file access, transfer, and remote file management** (rename, delete, list, etc.)
iii. Uses **single connection (Port 22)** unlike FTP
iv. Works well through firewalls
v. Commonly used in Linux/Unix environments

Example:
Transferring log files securely from a remote Linux server to a local machine

FTPS (FTP Secure):

i. FTP enhanced with **SSL/TLS encryption**
ii. Supports two modes:
    - **Explicit FTPS** (client requests security)
    - **Implicit FTPS** (secure connection required from start)
iii. Requires **digital certificates** for authentication
iv. Uses multiple ports (like FTP), making firewall configuration more complex

SCP (Secure Copy Protocol):

i. Uses Secure Shell for **secure file transfer**
ii. Simple, fast, and command-line based
iii. Only supports **file copying** (no browsing or management like SFTP)
iv. Ideal for quick transfers between systems

Key Differences Snapshot:

Feature	SFTP	FTPS	SCP
Encryption	SSH	SSL/TLS	SSH
Port	22	21 + others	22
Functionality	Full file management	FTP-like	Copy only
Firewall Friendly	Yes	No (complex)	Yes

Supporting Security Concepts:

i. **Encryption:** Converts data into unreadable format during transmission
ii. **Authentication Methods:**
    - Password-based
    - Key-based (SSH keys , more secure)
iii. **Integrity Checking:** Ensures files are not altered during transfer

Advantages:

i. Protects sensitive data from interception

ii. Ensures confidentiality and integrity
iii. Suitable for remote administration and cloud environments

Disadvantages:

i. Slight performance overhead due to encryption
ii. FTPS configuration can be complex
iii. SCP lacks flexibility

Real-World Use Cases:

i. Secure server backups
ii. Transferring confidential company data
iii. Managing cloud-hosted files
iv. DevOps deployments and automation pipelines

Security Best Practices:

i. Prefer **key-based authentication** over passwords
ii. Disable insecure protocols like plain FTP
iii. Regularly update SSH and SSL/TLS configurations
iv. Use strong encryption algorithms

Exam Tips:

i. **SFTP & SCP use SSH (Port 22)**
ii. **FTPS uses SSL/TLS (Port 21 + others)**
iii. SFTP = **full control**, SCP = **copy only**
iv. FTPS = **secure version of FTP but complex**

