UNIT 5: SECONDARY STORAGE MANAGEMENT AND I/O SYSTEM

1. Structure Overview of Mass Storage

- **Mass Storage:**
  Refers to **non-volatile** secondary storage devices used to store data permanently (e.g., Hard Drives, SSDs, Optical Disks).

Key Features:

- Large capacity
- Slower than main memory
- Persistent data (survives reboot)
- Examples: HDD, SSD, CD/DVD, Tape

2. Disk Structure

A. Physical Structure of Disk:

- **Platter:** Flat circular disk coated with magnetic material.
- **Track:** Concentric circles on the platter.
- **Sector:** Subdivision of a track; smallest addressable unit.
- **Cylinder:** Same track position on all platters.
- **Read/Write Head:** Reads and writes data.
- **Spindle:** Spins the platters at a constant speed (RPM).

B. Logical Structure:

- Logical Block Addressing (LBA):
  Abstracts the physical structure and assigns each block a unique number.

3. Disk Attachment

Types of Disk Interfaces:

1. **SATA (Serial ATA):**
   - Common in desktops/laptops.
   - Serial interface, moderate speed.
2. **SCSI (Small Computer System Interface):**
   - Older but fast; supports multiple devices.
3. **NVMe (Non-Volatile Memory Express):**
   - Used with SSDs; high speed via PCIe interface.
4. **USB Drives:**
   - Portable storage, slower than internal disks.

5. **RAID Arrays (Redundant Array of Independent Disks):**
    - Multiple disks used for redundancy and performance.

4. Disk Scheduling

- Goal: **Optimize disk head movement** and reduce seek time.

Common Scheduling Algorithms:

Algorithm	Description	Advantage	Drawback
FCFS	First-Come-First-Serve	Simple	Poor performance
SSTF	Shortest Seek Time First	Minimizes seek	May starve requests
SCAN	Elevator algorithm (head moves back and forth)	More fair	Skips edge requests sometimes
C-SCAN	Only services one direction, jumps back quickly	More uniform wait time	May waste head movement
LOOK / C-LOOK	Like SCAN/C-SCAN but stops at final request	Efficient movement	Slight complexity

Modern OS often use **LOOK-based variants**.

5. Disk Management

Involves preparing and organizing the disk for data storage.

Main Tasks:

- **Partitioning:**
    - Dividing the disk into logical sections.
    - Each partition can host a file system or swap space.
- **Formatting:**
    - Creating a file system on a partition.
    - Initializes metadata structures (e.g., FAT, ext4, NTFS).
- **Bad Block Management:**
    - Detects and avoids faulty sectors.
    - Low-level formatting or file system drivers handle it.
- **Volume Management:**
    - Tools like LVM (Linux Volume Manager) allow flexible volume resizing and spanning across multiple disks.

6. Swap-Space Management

- **Swap Space:**
  An area on disk used to **simulate additional RAM** by storing pages that are not currently used.

Why It's Needed:

- To support **virtual memory**.
- Helps when RAM is full and processes need more memory.

Methods of Allocation:

1. **Separate Swap Partition:**
   - Dedicated section on the disk.
   - Faster and managed by OS.
2. **Swap File:**
   - A file within the file system.
   - More flexible but slightly slower.

Management Strategies:

- **Preallocation:** Allocate swap at boot time.
- **Dynamic Allocation:** Allocate/deallocate as needed (used in some systems).

Performance Consideration:

- Swap access is much **slower than RAM**.
- Excessive swapping leads to **thrashing** (system slowdown).

Bonus: Performance Tips for Mass Storage

- Use **SSD** for OS and important files.
- Combine with **RAM caching** and **read-ahead** for speed.
- Apply **RAID** for redundancy and/or performance.
- Periodically **defragment** HDDs (not needed for SSDs).

RAID (Redundant Array of Independent Disks)

Purpose of RAID:

- Improve **performance** and/or **reliability** of data storage by using **multiple disks**.
- Offers **redundancy** to prevent data loss in case of disk failure.

RAID Levels and Comparison Table

RAID Level	Description	Performance	Fault Tolerance	Storage Efficiency
RAID 0	Striping only	High	None	100%
RAID 1	Mirroring	Read: High, Write: Moderate	High (1 disk can fail)	50%
RAID 2	Bit-level striping + Hamming ECC	Obsolete	High	Low
RAID 3	Byte-level striping + parity	Good for large files	High	~N−1/N
RAID 4	Block-level striping + dedicated parity	Good read	Moderate (parity bottleneck)	~N−1/N
RAID 5	Block-level striping + distributed parity	Balanced	Good (1 disk fault)	~N−1/N
RAID 6	Like RAID 5 + 2 parity blocks	Lower write speed	Excellent (2 disk fault)	~N−2/N
RAID 10 (1+0)	Striping across mirrors	High	Very High	50%

✅ **RAID 5** is commonly used in enterprise environments.
✅ **RAID 10** is preferred for high-performance and redundancy.

Storage Implementation

Storage Hierarchy:

1. **Registers** (fastest, smallest)
2. **Cache**
3. **Main Memory (RAM)**
4. **Secondary Storage** (HDD, SSD)
5. **Tertiary Storage** (Tape, Optical Disk)

Key Concepts in Implementation:

- **Block-Oriented Access:**
  - Storage is divided into fixed-size blocks (e.g., 4 KB).
  - File systems and virtual memory operate on blocks.
- **Caching:**
  - Recently/frequently accessed data is cached in RAM for speed.
- **Write Buffering:**
  - Write operations may be delayed to group them and reduce disk I/O.
- **Storage Stack:**

```
Application Layer
→ File System (e.g., ext4, NTFS)
→ Virtual File System (VFS)
→ Device Drivers
→ Disk Hardware
```

Tertiary Storage

Definition:

- Non-volatile storage used for **archiving** and **backup**.
- **Slower and cheaper** than primary or secondary storage.

Examples:

- Magnetic tapes
- Optical disks (CD/DVD/Blu-ray)
- Robotic storage systems (jukeboxes)

Characteristics:

Property	Value
Speed	Very slow access (minutes)
Cost	Very low per GB
Capacity	High (terabytes to petabytes)
Usage	Archival, backups, legal compliance

Access Types:

1. **Sequential Access:**
   - Data is accessed in a linear order (e.g., tapes).
2. **Automated Access:**
   - Robotic arms retrieve tapes/disks (used in large data centers).

Use Cases:

- Cold storage (data not accessed often)
- Backup solutions (e.g., disaster recovery)
- Data warehousing
- Compliance (long-term legal data retention)

1. Structure of I/O Systems

Purpose:

- Facilitate communication between **hardware devices** and **software applications**.
- Allow **efficient**, **secure**, and **abstracted** access to devices.

General Components:

1. **I/O Devices** (e.g., keyboard, disk, printer)
2. **Device Controllers** (hardware interface)
3. **Device Drivers** (software interface in the OS)
4. **Kernel I/O Subsystem**
5. **System Calls / Application Interface**

I/O Hardware

I/O Device Categories:

- **Block Devices**: Store data in fixed-size blocks (e.g., hard drives).
- **Character Devices**: Stream data byte-by-byte (e.g., keyboard, mouse).
- **Network Devices**: Transmit data packets (e.g., NICs).
- **Storage Devices**: USB drives, SSDs, CD/DVDs.

Key Hardware Components:

- **Device Controller**:
  - Interface between device and CPU.
  - Has **registers**, **buffer**, and **status information**.
- **Interrupts**:
  - Signal CPU to handle I/O completion.
  - Allows **asynchronous I/O**.
- **Direct Memory Access (DMA)**:
  - Transfers data between device and memory without CPU involvement.
  - Efficient for **bulk data transfers**.

Application I/O Interface

Role:

Provides a **standard API** for applications to access hardware without knowing device-specific details.

Types of I/O Access:

Access Type	Description
Blocking I/O	Process waits until I/O completes
Non-blocking I/O	Process continues and checks back later
Asynchronous I/O	Notified upon I/O completion (via signals or callbacks)
Memory-Mapped I/O	Maps device memory to address space (efficient access)

File Abstraction:

- Devices are treated like files in UNIX/Linux.
- Example: `/dev/sda` for a hard drive.

Kernel I/O Subsystem

Handles the **internal processing of I/O** within the OS kernel.

Key Responsibilities:

1. **Scheduling I/O Requests**:
    - Prioritize I/O tasks to improve throughput and fairness.
2. **Buffering**:
    - Temporary storage area for data being transferred.
    - Helps manage speed mismatches between devices and CPU.
3. **Caching**:
    - Stores recently accessed data for faster access.
    - Examples: Disk cache, read-ahead.
4. **Spooling**:
    - Queue output data (e.g., for printers) to be processed later.
5. **Device Independence**:
    - Uniform interface regardless of hardware type.
    - Achieved via device drivers.
6. **Error Handling**:
    - Detect and recover from hardware errors.

Transforming I/O Requests to Hardware Operations

This is the process by which a user-level I/O request becomes a physical hardware operation.

Step-by-Step Process:

1. **Application Calls a System Call**
   - Example: `read()`, `write()`, `open()`.
2. **OS Passes Request to Kernel I/O Subsystem**
   - It determines the right **device driver** and **device**.
3. **Device Driver Translates Request**
   - Converts the high-level operation to low-level **device-specific commands**.
4. **I/O Command Sent to Device Controller**
   - Via **bus** or **memory-mapped I/O**.
5. **Device Performs Operation**
   - e.g., Reads from disk, sends data over network.
6. **Interrupt or DMA Signals Completion**
   - OS resumes or notifies the requesting process.
7. **Data Transferred to User Space**
   - If applicable, buffered or cached first.

Example Flow: Reading a File from Disk

```
User Process
   ↓   (read())
Kernel I/O Subsystem
   ↓
File System Layer → Caching/Buffering
   ↓
Device Driver (Disk Driver)
   ↓
Disk Controller
   ↓
Disk Hardware
   ↓ (DMA Transfer)
Memory (RAM)
```