**UNIT 4: VIRTUAL MEMORY MANAGEMENT AND FILE MANAGEMENT**

## 1. Demand Paging

- **Definition:**
  A technique where pages are only loaded into memory **when they are needed**, rather than loading the entire process at once.
- **How It Works:**
  - Initially, no pages are loaded.
  - On a **page fault**, the OS loads the required page from disk to RAM.
  - Reduces memory usage and allows more processes to fit in memory.
- **Advantages:**
  - Less I/O.
  - Faster program startup.
  - Efficient memory usage.
- **Disadvantages:**
  - Page faults can cause delays.
  - If used poorly, can lead to **thrashing**.

## 2. Copy-On-Write (COW)

- **Definition:**
  A resource-management strategy where multiple processes share the same memory page until one needs to modify it.
- **How It Works:**
  - Commonly used during `fork()` in UNIX-like OSes.
  - Parent and child share memory.
  - On a write attempt, the OS **copies the page**, and each process gets its own version.
- **Benefits:**
  - Saves memory when pages are not modified.
  - Improves performance by avoiding unnecessary copying.
- **Use Cases:**
  - Process creation.
  - Virtual memory optimization.
  - Forking in multiprocessing environments.

## 3. Page Replacement

- **Definition:**
  When memory is full and a page fault occurs, the OS must **replace** a page in memory with a new one from disk.

- **Common Algorithms:**
    - **FIFO (First-In, First-Out):** Oldest page is replaced.
    - **LRU (Least Recently Used):** Page not used for the longest time is replaced.
    - **Optimal:** Replaces the page that will not be used for the longest time (theoretical, used for benchmarking).
    - **Clock Algorithm (Second-Chance):** Cycles through pages to find a suitable one to replace.
- **Page Fault Rate:**
    - Important to minimize.
    - Affected by algorithm choice and number of allocated frames.

### 4. Allocation of Frames

- **Definition:**
  Refers to how many frames (physical memory pages) are allocated to each process.
- **Types of Allocation:**
    - **Equal Allocation:** Each process gets the same number of frames.
    - **Proportional Allocation:** Based on the size or priority of processes.
    - **Priority-Based Allocation:** High-priority processes get more frames.
- **Global vs. Local Allocation:**
    - **Global Replacement:** Process can take frames from other processes.
    - **Local Replacement:** Process can only replace its own pages.
- **Minimum Number of Frames:**
    - Determined by hardware and the instruction set architecture (e.g., how many pages a single instruction can touch).

### 1. Thrashing

- **Definition:**
  **Thrashing** occurs when a system spends more time **swapping pages in and out of memory** than executing actual processes, due to frequent **page faults**.
- **Causes:**
    - High degree of multiprogramming.
    - Insufficient frames allocated to processes.
    - Processes exceeding their working sets (the set of pages they actively use).
- **Symptoms:**
    - High page-fault rate.
    - Low CPU utilization.
    - Very slow performance.
- **Solutions:**
    - **Reduce degree of multiprogramming** (kill or suspend some processes).
    - **Use working set model:** Allocate enough frames to each process based on its actual usage.
    - **Page fault frequency (PFF) control:** Monitor and adjust based on acceptable page fault rates.
    - **Use smarter page replacement algorithms** (like LRU instead of FIFO).

## 2. Memory-Mapped Files

- **Definition:**
  A technique that maps a file or a portion of a file into the **virtual memory address space** of a process.
- **How It Works:**
  - File contents appear in memory; the OS handles loading and saving transparently.
  - No need for explicit read/write system calls; memory access (load/store) is used instead.
  - Page faults are used to load file contents into memory as needed (lazy loading).
- **Advantages:**
  - Efficient I/O for large files.
  - Simplifies file access (especially in shared memory scenarios).
  - Shared memory between processes is easy (via shared file mappings).
- **Use Cases:**
  - Databases and multimedia applications.
  - Shared memory IPC (Inter-Process Communication).
  - Fast file access in operating systems and language runtimes.

## 3. Allocating Kernel Memory

- **Definition:**
  Refers to how memory is allocated **within the kernel space** for various kernel activities and data structures.
- **Requirements:**
  - Memory should be allocated quickly and must be **non-pageable** (always resident in RAM).
  - Must handle various allocation sizes (small for structures like process control blocks; large for buffers).
- **Allocation Methods:**
  1. **Buddy System:**
     - Allocates memory in power-of-2 sized blocks.
     - Fast coalescing and splitting.
     - Used for larger memory requests.
  2. **Slab Allocator:**
     - Efficient for allocating memory for objects of the same size (e.g., file descriptors, inodes).
     - Reduces fragmentation.
     - Uses caches (slabs) of pre-initialized objects.
- **Fragmentation Issues:**
  - **Internal Fragmentation:** Memory allocated but not used.
  - **External Fragmentation:** Free memory is in small pieces scattered across memory.
- **Kernel Memory Zones (in Linux):**
  - **ZONE_DMA:** For devices needing memory below a certain address.
  - **ZONE_NORMAL:** Regular memory.

    - **ZONE_HIGHMEM:** Memory not permanently mapped into the kernel space.

## 1. Other Considerations (in Memory Management)

These are additional memory management concerns not covered by primary mechanisms like paging or segmentation.

**Key Points:**

- **TLB (Translation Lookaside Buffer):**
    - Cache used to store recent virtual-to-physical address translations.
    - Speeds up memory access in paged systems.
- **Fragmentation:**
    - **Internal Fragmentation:** Wasted space within allocated memory blocks.
    - **External Fragmentation:** Free memory exists but is scattered.
- **Swapping:**
    - Moving entire processes in and out of memory to disk (less common today).
- **NUMA (Non-Uniform Memory Access):**
    - Memory access time varies depending on which processor accesses which part of memory.
- **Memory Protection:**
    - Ensures that processes cannot access each other's memory.
    - Implemented through hardware support like base-limit registers or page tables with protection bits.

## 2. Storage Management

- **Definition:**
  Refers to the efficient organization, storage, and retrieval of data on secondary storage devices like **hard disks** or **SSDs**.

**Key Responsibilities:**

- **Block Management:**
    - Data on disk is stored in fixed-size blocks.
    - Free blocks must be tracked (free lists, bitmaps).
- **Space Allocation Methods:**
    - **Contiguous:** Fast but causes fragmentation.
    - **Linked Allocation:** Easy to grow, but slow to access.
    - **Indexed Allocation:** Uses an index block to locate all data blocks.
- **Free Space Management:**
    - Techniques include bitmaps, free lists, and grouping of free blocks.
- **Caching and Buffering:**
    - Improve performance by keeping frequently used data in faster memory (RAM).
- **I/O Scheduling:**

    - Algorithms like FCFS, SSTF, SCAN, and LOOK optimize disk head movement to minimize latency.

## 3. File Concepts

- **Definition:**
  A **file** is a logical container for storing related data (e.g., text, images, executables).

**File Attributes:**

- Name
- Type
- Size
- Location
- Protection (permissions)
- Time and date of creation/modification/access

**File Operations:**

- Create
- Read/Write
- Delete
- Seek (move file pointer)
- Open/Close

**File Types:**

- Regular files (text, binary)
- Directories
- Special files (device files, pipes, sockets)

## 4. Access Methods

- **Definition:**
  Refers to how data in a file is read/written by programs.

**Types:**

1. **Sequential Access:**
    - Data is accessed in a linear, one-after-the-other fashion.
    - Most common (e.g., reading a text file).
    - Simple but not flexible.
2. **Direct (Random) Access:**
    - Data is accessed by specifying its location (offset).
    - Useful for databases and indexing.
    - Requires fixed-length records or block-based structure.

3. **Indexed Access:**
    - Uses an index (like in a book) to locate blocks or records.
    - Supports fast lookup and updates.
    - Common in modern file systems and databases.

## 1. Directory Structure

- **Definition:**
  A directory is a container that holds file entries. It organizes files hierarchically and allows efficient management and access.

**Common Directory Structures:**

1. **Single-Level Directory:**
    - All files are in one directory.
    - Easy to implement but causes name conflicts.
2. **Two-Level Directory:**
    - Each user has their own directory.
    - Prevents name conflicts but limits grouping flexibility.
3. **Tree-Structured Directory:**
    - Hierarchical structure (like Windows/Linux).
    - Allows file grouping and subdirectories.
    - Supports absolute and relative paths.
4. **Acyclic Graph Directory:**
    - Files/directories can have multiple parents using links.
    - Allows shared files but needs loop prevention.
5. **General Graph Directory:**
    - Like acyclic, but allows cycles.
    - Needs garbage collection to handle deleted files.

## 2. File System Mounting

- **Definition:**
  The process of making a file system accessible by attaching it to the main file system hierarchy.

**Key Concepts:**

- **Mount Point:**
  A directory where the new file system appears (e.g., `/mnt/usb` in Linux).
- **Mount Table:**
  Keeps track of all mounted file systems.
- **Unmounting:**
  Detaching the file system to prevent data corruption or loss.
- **Verification:**
  Before mounting, the OS verifies file system integrity and compatibility.

## 3. File Sharing

- **Definition:**
  Allows multiple users or processes to access the same file, either concurrently or over time.

**Mechanisms:**

- **User IDs and Groups:**
  - File permissions are managed based on ownership.
- **Shared File Pointers:**
  - Allows processes to share the same pointer (useful in multiprocessing).
- **Remote File Sharing:**
  - Using protocols like **NFS (Network File System)** or **SMB/CIFS (Windows)**.
- **Concurrency Control:**
  - Locks (advisory/mandatory) are used to prevent race conditions and inconsistencies.

## 4. Protection

- **Definition:**
  Ensures that only authorized users can access or modify files.

**Common Techniques:**

- **Access Control Lists (ACLs):**
  - Lists permissions for each user or group.
- **Unix File Permissions:**
  - Three sets: owner, group, others.
  - Permissions: read (r), write (w), execute (x).
- **Capabilities:**
  - Tokens or keys that grant specific access rights.
- **Encryption:**
  - Files can be encrypted to protect content even if accessed.
- **User Authentication:**
  - Ensures access is granted only to verified users.

## 5. Implementing a File System

- **Overview:**
  Involves creating structures on disk that support file and directory management.

**Core Components:**

- **Boot Control Block:**
  - Contains info to boot the system from the disk (if bootable).

- **Superblock (or File System Control Block):**
  - Stores file system type, size, and metadata like inode count, block size, etc.
- **Inode (Index Node):**
  - Metadata structure in Unix-like systems. Contains info like permissions, timestamps, and block pointers.
- **Directory Structure:**
  - Stores mappings between filenames and inodes.
- **File Allocation Table (FAT):**
  - Used in FAT file systems (DOS/Windows). Points to the next block of the file.
- **Journaling:**
  - Helps recover from crashes by logging metadata updates before applying them.

## 6. File System Structure

- **Physical Layout on Disk:**

**Components:**

1. **Boot Block:**
   - Contains bootloader code.
2. **Superblock:**
   - Key metadata about the file system.
3. **Inode Table:**
   - List of inodes for all files.
4. **Data Blocks:**
   - Actual contents of files.
5. **Free Space Management:**
   - Bitmaps or lists of unused blocks.

**Allocation Strategies:**

- **Contiguous Allocation:**
  - Fast access, but prone to fragmentation.
- **Linked Allocation:**
  - Each block contains a pointer to the next. No fragmentation but slower access.
- **Indexed Allocation:**
  - Uses index blocks to store all block addresses.

## 1. File System Structure Implementation

- **Definition:**
  The way the operating system organizes files and directories on **secondary storage** (like HDDs or SSDs).

**Key Components on Disk:**

1. **Boot Block:**
   - Contains OS loader (if disk is bootable).
2. **Superblock (File System Control Block):**
   - Holds metadata about the file system: total size, block size, number of inodes, etc.
3. **Inode Table / FAT Table:**
   - Stores information about individual files (metadata like size, owner, permissions, and data block pointers).
4. **Directory Structure:**
   - Maps file names to inodes or file descriptors.
5. **Data Blocks:**
   - Where the actual contents of files are stored.
6. **Free Space Map:**
   - Keeps track of which blocks are free and available for use.

## 2. Directory Implementation

- **Goal:**
  Efficiently map file names to inode numbers or file descriptors and support fast lookups, creation, and deletion.

**Two Common Methods:**

1. **Linear List:**
   - Directory is a simple list of file entries.
   - Easy to implement but **slow for large directories** (O(n) lookup).
2. **Hash Table:**
   - Hashes file names to entries.
   - Much **faster lookup (O(1) on average)**.
   - Requires extra memory and hash collision handling.

**Directory Entry Contains:**

- File name
- File type
- Inode number (or pointer to metadata)
- Possibly timestamps and size

## 3. Allocation Methods

How the file system allocates **disk blocks** to files.

**1. Contiguous Allocation:**

- **Files stored in sequential blocks.**
- Fast access (especially for sequential reads).
- **Drawbacks:**
    - External fragmentation.
    - File size must be known in advance.

**2. Linked Allocation:**

- Each file block contains a pointer to the next block.
- No fragmentation.
- **Drawbacks:**
    - Random access is slow (O(n)).
    - Extra space used for pointers.

**3. Indexed Allocation:**

- Uses a special index block that contains all the pointers to the file's blocks.
- Supports direct access and large files.
- **Example:** Inodes in UNIX/Linux.
- **Drawbacks:**
    - Index block can be limited in size unless multi-level indexing is used.

## 4. Free Space Management

How the OS keeps track of which blocks are free and available for new data.

**Common Techniques:**

1. **Bit Map (Bit Vector):**
    - 1 = block used, 0 = block free.
    - Very compact, easy to check contiguous space.
    - Can be slow to find large free blocks.
2. **Free List:**
    - Linked list of free blocks.
    - Simple to implement, fast for allocating single blocks.
    - Not efficient for allocating contiguous blocks.
3. **Grouping:**
    - Stores addresses of free blocks in groups.
    - Improves space utilization and reduces search time.

4. **Counting:**
    - Keeps track of free blocks in runs (e.g., start block + count).
    - Good for systems with long runs of free space.

## 5. Efficiency and Performance

The goal is to optimize speed, minimize overhead, and make effective use of storage.

**Techniques to Improve Performance:**

- **Buffer Cache (Disk Cache):**
    - Frequently accessed disk blocks are kept in memory.
    - Reduces I/O access times.
- **Read-Ahead:**
    - Predict and preload blocks into memory before they are requested.
- **Write-Back vs. Write-Through:**
    - Write-back: Writes are cached, improving performance but less safe.
    - Write-through: Writes go directly to disk, safer but slower.
- **Clustering:**
    - Group blocks together to reduce seek time.
- **Defragmentation:**
    - Rearranges blocks on disk to reduce fragmentation and improve access speed.
- **Block Size Tuning:**
    - Small blocks reduce internal fragmentation but may increase overhead.
    - Large blocks are more efficient for large files but waste space for small files.

## 1. What is Recovery?

- **Definition:**
  Recovery is the process of **restoring a consistent state** of the system after a failure.
  It's especially important for **file systems**, **databases**, and **transactional systems**.
- **Objective:**
    - Ensure **data integrity** and **consistency**.
    - Resume normal operation after:
        - System crash
        - Power failure
        - Disk failure
        - File corruption

## 2. Types of Failures

1. **System Crash (Volatile Memory Loss):**
    - OS crashes or power outage.
    - Main memory (RAM) contents are lost.
    - Disk content remains intact.
2. **Disk Failure:**
    - Physical disk damage or data corruption.
    - Can result in permanent data loss.
3. **Software Errors:**
    - Bugs or improper shutdowns may corrupt the file system or metadata.
4. **Transaction Failures:**
    - Interruptions in multi-step operations (common in databases).

## 3. Recovery Techniques

### A. Consistency Checking (fsck, chkdsk):

- Utility scans the file system and repairs inconsistencies.
- Example:
    - In UNIX/Linux: `fsck`
    - In Windows: `chkdsk`

### B. Backups:

- Periodic copying of files or entire file systems to secondary storage.
- Allows manual or automatic restoration.

### C. Journaling (Log-Based Recovery):

- Used in **journaling file systems** (e.g., ext3/ext4, NTFS).
- All metadata updates are first written to a **journal (log)**.
- In case of a crash:
    - The journal is replayed to recover the correct state.
- **Two phases:**

1. **Write-Ahead Logging (WAL):** Log before writing to disk.
2. **Commit or Abort:** Apply changes only if the transaction is complete.

### D. Shadow Paging:

- Used in databases and advanced file systems.
- A copy (shadow) of the original page is made before modification.
- If failure occurs, original remains safe.

## 4. Steps in Recovery Process

1. **Detection of Failure:**
   - Through system logs, crash reports, or inconsistency flags.
2. **Analysis:**
   - Determine the state before the crash (via logs or snapshots).
3. **Restore Consistency:**
   - Apply recovery tools (like journaling replay or fsck).
   - Restore files from backup if needed.
4. **Validation:**
   - Ensure recovered file system is working and consistent.

## 5. File System Specific Recovery Support

| File System | Recovery Feature |
|---|---|
| **ext3/ext4** | Journaling (metadata or full) |
| **NTFS** | Journaling and Transaction Logs |
| **XFS** | Metadata journaling |
| **ZFS** | Copy-on-write & checksums |
| **FAT32** | No built-in recovery (requires chkdsk) |

## 6. Best Practices for Recovery

- Use **journaling file systems** to reduce recovery time.
- Keep **regular backups** (incremental & full).
- Use **UPS systems** to prevent data loss from power failures.
- Automate **periodic file system checks**.