**Objectives of an Operating System**

1. **Convenience**
   * Provides a user-friendly interface (GUI/CLI) for easy interaction with the computer.
2. **Efficiency**
   * Maximizes resource utilization (CPU, memory, storage, etc.) for optimal performance.
3. **Reliability**
   * Ensures stable and error-free operation, preventing system crashes.
4. **Scalability**
   * Supports hardware upgrades and software expansion without major changes.
5. **Security & Privacy**
   * Protects data from unauthorized access and malware.
6. **Portability**
   * Can run on different hardware platforms with minimal modifications.

**Functions of an Operating System**

The OS performs several key functions, categorized as follows:

**1. Process Management**

* **Process Scheduling:** Allocates CPU time using algorithms (FCFS, Round Robin, Priority Scheduling).
* **Multitasking:** Allows multiple processes to run concurrently.
* **Interprocess Communication (IPC):** Enables processes to exchange data.
* **Deadlock Handling:** Detects and resolves deadlocks (if they occur).

**2. Memory Management**

* **Allocation/Deallocation:** Assigns memory to processes and reclaims it when done.
* **Virtual Memory:** Uses paging & swapping to extend usable memory.
* **Protection:** Prevents processes from accessing unauthorized memory.

**3. File System Management**

* **File Creation/Deletion:** Manages files and directories.
* **Disk Space Management:** Allocates storage efficiently.
* **Access Control:** Enforces file permissions (read/write/execute).

**4. Device Management (I/O Management)**

* **Device Drivers:** Interfaces with hardware (printers, keyboards, etc.).
* **Buffering & Caching:** Improves I/O performance.
* **Plug & Play:** Detects and configures new hardware automatically.

**5. Security & Access Control**

* **User Authentication:** Verifies user identity (passwords, biometrics).
* **Encryption:** Protects sensitive data.
* **Firewall & Antivirus Integration:** Blocks malicious activities.

**6. Networking & Communication**

* **Network Protocols:** Supports TCP/IP, HTTP, FTP, etc.
* **Remote Access:** Enables SSH, RDP, and VPN connections.

**7. User Interface (UI)**

* **GUI (Graphical UI):** Windows, icons, menus (e.g., Windows, macOS).
* **CLI (Command Line):** Text-based commands (e.g., Linux Terminal, CMD).

**8. System Performance Monitoring**

* **Task Manager/Performance Logs:** Tracks CPU, RAM, and disk usage.
* **Error Detection & Logging:** Identifies and logs system errors.

**RAG**

A **Resource Allocation Graph (RAG)** is a directed graph used to model resource allocation and deadlock detection in an operating system. It visually represents processes, resources, and their relationships, helping identify potential deadlocks.

**Components of RAG**

1. **Process Nodes (P₁, P₂, ...)**
   * Represented by **circles (○)**.
   * Indicate active processes in the system.
2. **Resource Nodes (R₁, R₂, ...)**
   * Represented by **rectangles (□)**.
   * Can be single-instance (one unit) or multi-instance (multiple units).
   * **Dots (•)** inside a resource node represent available instances.
3. **Edges (Directed Arrows →)**
   * **Request Edge (P → R):** Process is **waiting** for the resource.
   * **Assignment Edge (R → P):** Resource is **allocated** to the process.

**Types of RAGs**

1. **Single-Instance Resources (No Multiple Units)**
   * Each resource has only **one unit**.
   * A cycle in the graph **always** means a deadlock.
2. **Multi-Instance Resources (Multiple Units Available)**
   * A resource can have **multiple instances** (e.g., 3 printers).
   * A cycle **does not always** mean a deadlock (some instances may still be free).

Deadlock Detection Using RAG

* **If the graph has no cycle → No deadlock.**
* **If the graph has a cycle:**
  + **For single-instance resources → Deadlock exists.**
  + **For multi-instance resources → Check using Banker’s Algorithm.**

## **File Attributes (File Metadata)**

File attributes are properties that describe a file and help the OS manage it. Common attributes include:

|  |  |
| --- | --- |
| ****Attribute**** | ****Description**** |
| **Name** | Unique identifier (e.g., document.txt). |
| **Type** | Extension indicating file format (e.g., .txt, .exe). |
| **Location** | Path where the file is stored (e.g., C:\Users\File.doc). |
| **Size** | Storage occupied (in bytes, KB, MB, etc.). |
| **Creation Time** | When the file was created. |
| **Last Modified** | When the file was last edited. |
| **Last Accessed** | When the file was last opened. |
| **Owner** | User who created the file (important for permissions). |
| **Permissions** | Read/Write/Execute access for users/groups. |

## **File Operations**

The OS provides various operations to manage files:

|  |  |
| --- | --- |
| **Operation** | **Description** |
| **Create** | Allocates storage and registers the file. |
| **Open** | Loads file metadata for access. |
| **Read** | Retrieves data from the file. |
| **Write** | Modifies or appends data to the file. |
| **Seek** | Moves the file pointer to a specific location. |
| **Delete** | Removes the file and frees storage. |
| **Truncate** | Erases contents but keeps the file. |
| **Rename** | Changes the file name. |
| **Copy** | Duplicates the file to another location. |
| **Move** | Relocates the file (cut + paste). |
| **Get Attributes** | Retrieves metadata (size, permissions, etc.). |
| **Set Attributes** | Modifies permissions, ownership, etc. |

**Virtual memory**

is a memory management capability of an operating system (OS) that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage. Virtual memory is the separation of user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available, or about what code can be placed in overlays, but can concentrate instead on the problem to be programmed.

Advantages:

## **1. Enables Efficient Multitasking**

* Allows multiple programs to run simultaneously even if RAM is insufficient.
* Each process gets its own virtual address space, preventing interference.

## **2. Increases Effective Memory Capacity**

* Combines RAM and disk space to provide the illusion of a larger memory.
* Lets systems run applications that require more memory than physically available.

## **3. Simplifies Memory Management**

* Programs use **logical (virtual) addresses**, while the OS maps them to physical RAM/disk.
* Developers don’t need to worry about physical memory constraints.

## **4. Enhances System Stability & Security**

* Prevents applications from crashing due to "Out of Memory" errors.
* Isolates processes, preventing one program from accessing another’s memory.

## **5. Supports Larger Applications**

* Enables execution of programs larger than available RAM (e.g., video editing, gaming).
* Uses **paging/swapping** to load only required parts into RAM.

## **6. Optimizes RAM Usage (Demand Paging)**

* Loads only the necessary pages into RAM, reducing wastage.
* Unused pages stay on disk until needed (**lazy loading**).

## **7. Facilitates Memory Sharing**

* Multiple processes can share the same code (e.g., libraries like DLL files).
* Reduces duplication, saving RAM.

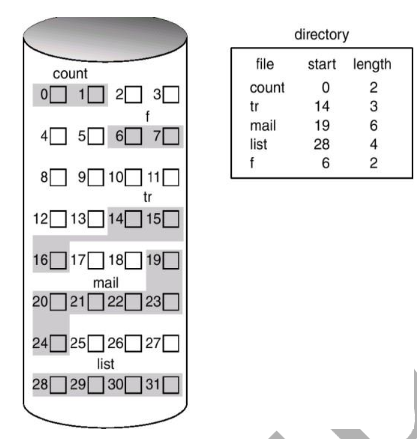
### ****Different File Allocation Methods in Operating Systems****

File allocation methods define how files are stored on disk. The three main techniques are:

## **1. Contiguous Allocation:** In contiguous allocation, each file occupies a set of consecutive disk blocks. The OS stores the starting block address and length of the file. This method ensures fast access since the entire file is stored sequentially, reducing seek time. It works well for pre-allocated storage but suffers from external fragmentation over time. Files cannot expand easily unless adjacent space is free, leading to inefficient storage use. Defragmentation is often required to reclaim fragmented space.

**Advantages:**  
 Fast sequential and random access.  
 Minimal disk head movement.  
 Simple to implement.

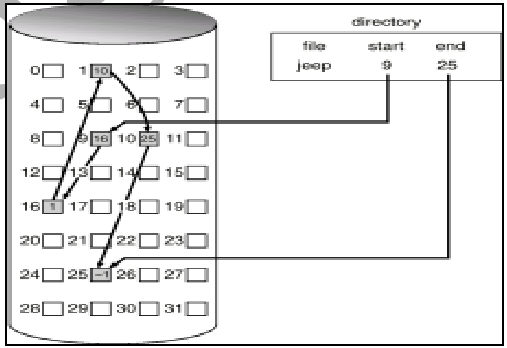
**Disadvantages:**  
External fragmentation occurs.  
Difficult to grow files dynamically.  
Requires pre-allocation of space.



## **2. Linked Allocation:** Linked allocation stores files as linked lists of disk blocks, where each block contains a pointer to the next block. The directory entry holds the first and last block addresses. This eliminates external fragmentation since blocks need not be contiguous. However, random access is slow because traversal is required. Another issue is reliability—if a pointer is corrupted, the entire file may become inaccessible. File Allocation Table (FAT) is a variation that improves performance by storing pointers in a separate table.

**Advantages:**  
 No external fragmentation.  
 Efficient for dynamic file growth.  
 Simple to manage free space.

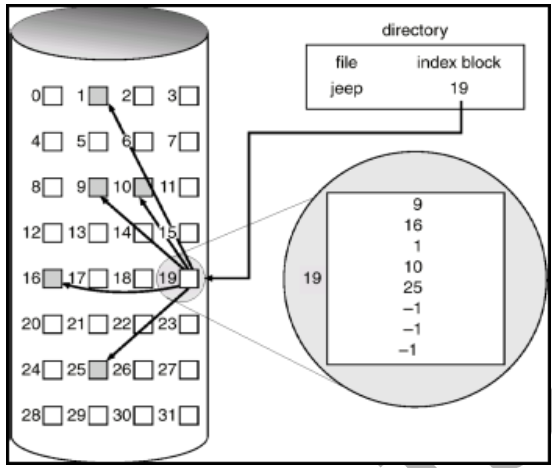
**Disadvantages:**  
Poor random access performance.  
Overhead due to pointer storage.  
Less reliable (broken links lose data).



## **3. Indexed Allocation:** Indexed allocation uses an index block containing pointers to all disk blocks of a file. The directory entry points to this index block, allowing fast direct access to any part of the file. This method eliminates external fragmentation and supports dynamic file growth efficiently. However, storing index blocks consumes extra space, and large files may need multiple-level indexing (e.g., UNIX inodes). It balances speed and flexibility but increases complexity compared to contiguous or linked allocation.

**Advantages:**  
 Supports fast random access.  
 No external fragmentation.  
 Flexible for dynamic file sizes.

**Disadvantages:**  
Extra overhead for index storage.  
Slower for very small files.  
Complex implementation with multi-level indexing.

. 

**Deadlock**

A deadlock is a situation in an operating system where two or more processes are blocked forever, each waiting for a resource held by the other.

1. **Mutual Exclusion**
   * At least one resource must be held in a non-sharable mode (only one process can use it at a time).
   * Example: A printer cannot be shared while printing.
2. **Hold and Wait**
   * A process must hold at least one resource while waiting to acquire additional resources held by other processes.
   * Example: Process A holds a file and waits for a scanner held by Process B.
3. **No Preemption**
   * Resources cannot be forcibly taken away; they must be voluntarily released by the process holding them.
   * Example: A process holding a locked file cannot be interrupted to release it.
4. **Circular Wait**
   * A circular chain of processes exists where each process waits for a resource held by the next process in the chain.
   * Example: Process A waits for Process B, which waits for Process C, which waits for Process A.

**Internal fragmentation**

Occurs when there is unused space within a memory block. For example, if a system allocates a 64KB block of memory to store a file that is only 40KB in size, that block will contain 24KB of internal fragmentation. When the system employs a fixed-size block allocation method, such as a memory allocator with a fixed block size, this can occur.

**External fragmentation**

Occurs when a storage medium, such as a hard disc or solid-state drive, has many small blocks of free space scattered throughout it. This can happen when a system creates and deletes files frequently, leaving many small blocks of free space on the medium. When a system needs to store a new file, it may be unable to find a single contiguous block of free space large enough to store the file and must instead store the file in multiple smaller blocks. This can cause external fragmentation and performance problems when accessing the file.

* **Slower Performance:** Fragmentation slows down the read and write speed of the disk, as the disk head has to move to different locations to access the fragments of a file. This, in turn, increases the access time and reduces the overall speed of the system, causing slowdowns and lag in applications.
* **Disk Space Wasting:** Fragmentation can also lead to disk space being wasted, as fragments may occupy more space than required. This can result in a shortage of disk space, causing the system to become unstable and vulnerable to crashes or errors.
* **Data Loss:** In severe cases, fragmentation can also cause the system to run out of disk space, leading to data loss. This can be particularly detrimental to users who rely on their systems to store critical information and data.
* **Increased Risk of System Crashes:** The more fragmented a disk becomes, the more prone it is to crashes and errors. This can cause the system to become unstable and vulnerable to data loss, crashes, and other issues.
* **Reduced Battery Life:** Fragmentation can also negatively impact the battery life of laptops and other [mobile devices.](https://www.geeksforgeeks.org/challenge-in-mobile-device/) This is because fragmentation requires the disk to work harder, which in turn puts additional strain on the battery.

**Static and Fixed Partitions**

Static partitioning divides memory into fixed-size blocks at system startup. Each partition holds one process, and sizes remain unchanged. If a process is smaller than its partition, internal fragmentation occurs. This method is simple but inefficient for variable-sized processes.

**Advantages:**

* Fast allocation (no runtime calculations).
* No external fragmentation.
* Predictable memory usage.

**Disadvantages:**

* Internal fragmentation wastes memory.
* Inflexible (cannot adapt to process sizes).
* Poor utilization for dynamic workloads.

**Dynamic Partitions**

Dynamic partitioning allocates memory at runtime, creating partitions that match process sizes exactly. It eliminates internal fragmentation but causes external fragmentation over time as free memory becomes scattered.

**Advantages:**

* No internal fragmentation.
* Efficient memory use (fits exact process sizes).
* Flexible for varying workloads.

**Disadvantages:**

* External fragmentation requires compaction.
* Slower allocation (must search for free blocks).
* Complex management overhead.

**One-to-One Model**

**Explanation:** Each user thread maps directly to a kernel thread, enabling true parallelism. The OS manages these threads, allowing multiple threads to run simultaneously on multi-core CPUs. This model is common in modern OS like Windows and Linux.

**Advantages:**

* Supports true parallel execution.
* Blocking one thread does not affect others.
* Better utilization of multi-core processors.

**Disadvantages:**

* High overhead due to kernel involvement.
* Limited by OS constraints on thread count.
* Increased resource consumption.

**Many-to-One Model**

**Explanation:** Multiple user threads are mapped to a single kernel thread. Thread management occurs in user space, making it lightweight but incapable of true parallelism.

**Advantages:**

* Low overhead due to user-level management.
* Portable across different operating systems.
* Efficient for simple, non-CPU-bound tasks.

**Disadvantages:**

* No true parallelism (single kernel thread).
* Blocking one thread blocks all threads.
* Poor scalability on multi-core systems.

**Many-to-Many Model**

**Explanation:** User threads are multiplexed onto a pool of kernel threads, balancing parallelism and efficiency. This model avoids the limitations of one-to-one and many-to-one approaches.

**Advantages:**

* Supports true parallelism without excessive overhead.
* Flexible thread allocation (scalable).
* Efficient for high-performance applications.

**Disadvantages:**

* Complex implementation.
* Requires careful management of thread pools.
* Potential for underutilization if not configured properly.

**One-to-Many Model**

**Explanation:** A single user thread maps to multiple kernel threads, though this model is rare due to its impracticality in most scenarios.

**Advantages:**

* Theoretical potential for enhanced parallelism.
* Could optimize specific niche workloads.

**Disadvantages:**

* Uncommon and poorly supported in modern OS.
* High complexity with minimal benefits.
* Inefficient for general-purpose computing.

**Paging**

Paging is a memory management technique that divides the physical memory into fixed-sized blocks called as frames. Paging eliminates external fragmentation by dividing physical memory into fixed-size blocks called frames and logical memory into same-sized pages. The OS maintains a page table to map logical pages to physical frames.

**Advantages of Paging**

* Eliminates external fragmentation by using fixed-size pages.
* Enables efficient memory allocation for non-contiguous processes.
* Simplifies memory management with uniform page sizes.

**Disadvantages of Paging**

* Internal fragmentation may occur if process size is not a multiple of page size.
* Overhead due to page table storage and lookups.
* Page faults can degrade performance if disk access is frequent.

**Content of Page Table:**

* Page Number
* Page Offset
* Frame Number
* Protection Bit
* Present/Absent Bit
* Dirty Bit
* Reference Bit

RAID

RAID (Redundant Array of Independent Disks) is like having backup copies of your important files stored in different places on several hard drives or solid-state drives (SSDs). If one drive stops working, your data is still safe because you have other copies stored on the other drives. It's like having a safety net to protect your files from being lost if one of your drives breaks down.

**RAID (Redundant Array of Independent Disks) Explained**

**RAID 0 (Striping)**

**Description:** Data is split evenly across two or more disks with no redundancy.  
**Advantages:**

1. Fast read/write speeds due to parallel I/O.
2. Full storage capacity utilization.
3. Simple implementation.  
   **Disadvantages:**
4. No fault tolerance—failure of one disk loses all data.
5. Not suitable for critical data.
6. Higher risk of data loss.

**RAID 1 (Mirroring)**

**Description:** Data is duplicated on two or more disks for redundancy.  
**Advantages:**

1. High fault tolerance—one disk failure doesn’t lose data.
2. Fast read performance (reads from both disks).
3. Simple recovery process.  
   **Disadvantages:**
4. 50% storage efficiency (half capacity wasted).
5. Slower write speeds (data written twice).
6. Expensive due to disk duplication.

**RAID 5 (Striping with Parity)**

**Description:** Data and parity are striped across three or more disks, allowing single-disk failure recovery.  
**Advantages:**

1. Balanced performance and redundancy.
2. Efficient storage (only one disk’s worth lost to parity).
3. Supports single-disk failure recovery.  
   **Disadvantages:**
4. Slow writes due to parity calculation.
5. Rebuilding array after failure is slow.
6. Not ideal for write-heavy workloads.

**RAID 6 (Striping with Double Parity)**

**Description:** Like RAID 5 but with two parity blocks, tolerating two disk failures.  
**Advantages:**

1. Higher fault tolerance (two disk failures allowed).
2. Good for large, critical storage systems.
3. Better reliability than RAID 5.  
   **Disadvantages:**
4. More storage overhead (two disks for parity).
5. Slower write performance than RAID 5.
6. Complex rebuild process.

**RAID 10 (1+0: Mirroring + Striping)**

**Description:** Combines RAID 1 (mirroring) and RAID 0 (striping) for speed and redundancy.  
**Advantages:**

1. High performance (striping) + redundancy (mirroring).
2. Faster rebuilds than RAID 5/6.
3. Supports multiple disk failures (if in different mirrors).  
   **Disadvantages:**
4. 50% storage efficiency (like RAID 1).
5. Expensive (requires at least four disks).
6. Not scalable for very large arrays.

Design Issues of Mobile and Real-Time Operating Systems

1. Mobile Operating System (Mobile OS) Design Issues

A Mobile OS is designed for smartphones, tablets, and embedded devices. Key design challenges include:

Power Management: Mobile devices run on batteries, so efficient power consumption is crucial.

Limited Resources: Memory, CPU, and storage are limited compared to desktop OS.

User Interface (UI): Should support touch screens, gestures, and voice commands.

Connectivity: The system should manage Wi-Fi, Bluetooth, 4G/5G, and GPS efficiently.

Security & Privacy: Protection from malware, app permissions, and secure authentication are essential.

App Management: Should support multitasking and background processes efficiently.

Real-Time Updates: Frequent updates are required for security and performance improvements.

**Real-Time Operating System (RTOS) Design Issues**

A Real-Time OS is designed to handle time-critical tasks where system response time is crucial. It is used in industrial automation,

robotics, avionics, and medical devices.

Deterministic Response Time: RTOS must process tasks within a strict deadline.

Task Scheduling: Needs efficient scheduling to manage real-time tasks.

Interrupt Handling: Quick and predictable response to external events is essential.

Concurrency & Synchronization: Must manage multiple tasks and shared resources without delays.

Reliability & Fault Tolerance: The system must operate continuously without failures.

Minimal Latency: Ensures the shortest response time for real-time operations.

Memory Management: Predictable and static memory allocation to prevent delays

|  |  |  |
| --- | --- | --- |
| **FACTOR** | **PROCESS SCHEDULING** | **PROCESS SWITCHING (CONTEXT SWITCHING)** |
| PURPOSE | DECIDES WHICH PROCESS GETS CPU TIME NEXT (POLICY). | EXECUTES THE TRANSITION BETWEEN PROCESSES BY SAVING/LOADING STATES (MECHANISM). |
| WHEN IT OCCURS | TRIGGERED BY EVENTS (E.G., TIME SLICE EXPIRY, I/O REQUEST, PRIORITY CHANGE). | OCCURS AFTER SCHEDULING TO IMPLEMENT THE DECISION (E.G., WITCH TO THE SELECTED PROCESS). |
| MAIN OPERATIONS | ALGORITHM-BASED SELECTION | SAVING THE CURRENT PROCESS’S STATE AND LOADING THE NEXT PROCESS’S STATE (VIA PCB). |
| KEY COMPONENTS | SCHEDULER (ALGORITHM), READY QUEUE, PRIORITY SYSTEM. | PROCESS CONTROL BLOCK (PCB), KERNEL MODE OPERATIONS, CPU REGISTERS. |
| TRIGGERS | TIMER INTERRUPTS, PROCESS YIELDING, OR NEW HIGH-PRIORITY PROCESS ARRIVAL. | SCHEDULER’S DECISION, HARDWARE INTERRUPTS (E.G., I/O COMPLETION), OR SYSTEM CALLS. |
| OVERHEAD | LOW | HIGH |
| DEPENDENCY | CAN OCCUR WITHOUT A CONTEXT SWITCH | ALWAYS FOLLOWS SCHEDULING |
| EXAMPLES | ALGORITHMS: FCFS, SJF, ROUND ROBIN, PRIORITY SCHEDULING. | STEPS: SAVE CPU STATE OF PROCESS A → LOAD CPU STATE OF PROCESS B → UPDATE PC AND REGISTERS. |