1. Basics of OS

1. What is an Operating System?
   An Operating System (OS) is a fundamental software that acts as an intermediary between the user and the computer hardware. Its primary role is to manage all the software and hardware resources of the computer system, providing a stable and consistent environment for applications to run and for users to interact with the machine. Think of it as the "conductor of an orchestra," where the conductor (OS) makes sure all the musicians (hardware components) play together harmoniously to produce a beautiful piece of music (the user's work).
2. What are the main functions of an OS?
   The main functions of an OS include:
   1. **Process Management:** It creates, schedules, manages, and terminates processes.
   2. **Memory Management:** It allocates and deallocates memory to different processes to prevent them from interfering with each other.
   3. **File Management:** It organizes files and directories, handling file creation, deletion, and access.
   4. **Device Management:** It controls and communicates with hardware devices (e.g., keyboard, mouse, printer).
   5. **Security & Protection:** It provides mechanisms to protect system resources and data from unauthorized access.
   6. **User Interface:** It provides a way for users to interact with the computer, either through a Graphical User Interface (GUI) or a Command Line Interface (CLI).
3. **Types of Operating Systems (Batch, Time-sharing, Real-time, Distributed, etc.)**
   1. **Batch OS:** Executes jobs in batches without user interaction. Examples include payroll systems or large data processing jobs.
   2. **Time-sharing OS:** Allows multiple users to use the same computer simultaneously. The CPU is shared among multiple processes, each getting a "time slice." **Example:** Unix.
   3. **Real-time OS (RTOS):** Guarantees that a specific task will be completed within a fixed time frame. Used in systems where timeliness is critical. **Example:** Control systems for robots or medical imaging devices.
   4. **Distributed OS:** Manages a group of independent computers and makes them appear as a single system. **Example:** Google's search engine infrastructure.
   5. **Network OS:** Runs on a server and provides capabilities to manage data, users, groups, and security for a network. **Example:** Windows Server, Red Hat Enterprise Linux.
4. Difference between 32-bit and 64-bit OS.
   The main difference lies in how they handle memory and data.
   1. **32-bit OS:** Can access a maximum of 4 GB of RAM. The CPU processes data in 32-bit chunks.
   2. **64-bit OS:** Can access a theoretical maximum of over 16 exabytes of RAM. The CPU processes data in 64-bit chunks, making it faster for handling large files and complex calculations. Most modern systems use 64-bit OS.
5. **Difference between kernel and shell.**

    1. **Kernel:** The **core component of the OS**. It's the lowest-level part, directly interacting with the hardware. It handles essential tasks like memory management, process scheduling, and device I/O. The kernel acts as a bridge between applications and hardware.
    2. **Shell:** A **user interface** that provides a command-line interpreter. It takes commands from the user and translates them for the kernel to execute. **Example:** Bash, Zsh (in Linux/Unix) or Command Prompt (in Windows).
6. What is system call? Examples?
   A system call is a programmatic way for a computer program to request a service from the kernel of the operating system. It's the interface between a process and the OS.
   Examples:
    1. `fork()`: To create a new process.
    2. `read()`: To read data from a file or device.
    3. `write()`: To write data to a file or device.
    4. `exit()`: To terminate a process.
7. **Difference between monolithic kernel and microkernel.**
    1. **Monolithic Kernel:** All OS services (process management, memory management, file systems, device drivers) run in a single address space in kernel mode. It's fast because all components can directly communicate. **Example:** Linux, Unix.
    2. **Microkernel:** The kernel is kept as small as possible. It only contains essential services like inter-process communication (IPC) and basic memory management. Other services (like file systems and device drivers) run as separate user-space processes. It's more secure and reliable as a crash in one service won't bring down the entire system. **Example:** Minix, GNU Hurd.
8. **What is multitasking, multiprogramming, multithreading?**
    1. **Multitasking:** The ability of an OS to execute multiple tasks or processes **concurrently** over a period of time by rapidly switching between them. **Example:** You can browse the web while listening to music.
    2. **Multiprogramming:** The ability to execute multiple programs **simultaneously** on a single processor. The OS keeps multiple jobs in memory and executes them one after the other, to maximize CPU utilization. It's a subset of multitasking.
    3. **Multithreading:** The ability of a single process to execute multiple threads **concurrently**. Threads share the same memory space, making them "lightweight processes." **Example:** In a web browser, one thread handles user input, another downloads a file, and a third displays images.
9. **Difference between process and program.**
    1. **Program:** A **passive entity**; it's a file containing a set of instructions stored on disk. It's a static entity.
    2. **Process:** An **active entity**; it's an instance of a program that is being executed. It has its own memory space (stack, heap, data), program counter, and resources. When you click on an application's icon, you create a process.
10. What is booting process in OS?
    The booting process is the sequence of operations that a computer goes through when it starts up.

1. **BIOS/UEFI Initialization:** The computer's firmware (BIOS or UEFI) performs a Power-On Self Test (POST) to check hardware.
2. **Bootloader Execution:** The firmware loads the bootloader from a storage device (e.g., hard drive). The bootloader's job is to load the OS kernel. **Example:** GRUB in Linux.
3. **Kernel Loading & Initialization:** The bootloader loads the OS kernel into memory. The kernel then initializes essential data structures and services.
4. **Init Process Start:** The kernel starts the first process, which is often called the `init` process (or `systemd` in modern Linux). This process is responsible for starting all other system services and user processes.

---

2. Processes and Threads

11. What is a process? Different states of a process?
    A process is an instance of a program in execution. It's the unit of work in a system.
    A process can be in one of several states:
    - **New:** The process is being created.
    - **Ready:** The process is waiting to be assigned to a processor.
    - **Running:** Instructions are being executed by the CPU.
    - **Waiting/Blocked:** The process is waiting for some event to occur (e.g., I/O completion).
    - **Terminated/Completed:** The process has finished execution.
12. **Difference between process and thread.**
    - **Process:** A **heavyweight** entity. Processes have their own separate address space, so they are isolated from each other. Communication between processes is slower (via IPC). Context switching between processes is more expensive.
    - **Thread:** A **lightweight** entity. Threads within the same process share the same memory space, code, and data. Communication between threads is faster (via shared memory). Context switching between threads is less expensive.
13. What is PCB (Process Control Block)?
    A Process Control Block (PCB) is a data structure in the OS kernel that contains all the information about a process. The OS uses the PCB to manage and control a process.
    The PCB contains:
    - **Process State:** (New, Ready, Running, etc.)
    - **Process ID (PID):** A unique identifier for the process.
    - **Program Counter:** The address of the next instruction to be executed.
    - **CPU Registers:** Values of all CPU registers for the process.
    - **Memory Management Information:** Page tables or segment tables.
    - **I/O Status:** List of open files, I/O devices allocated.
14. What are context switches?
    A context switch is the process of saving the state of a currently running process and loading the saved state of another process to resume its execution. It allows the OS

to switch the CPU from one process to another, enabling multitasking.
Steps:
    - Save the state of the current process into its PCB.
    - Load the state of the new process from its PCB.
    - Update the CPU's program counter to point to the new process's next instruction.
      Context switches are a form of overhead as they don't perform any useful work.
15. **Difference between user-level and kernel-level threads.**
    - **User-level Threads:** Managed by a user-level library without kernel support. The kernel is unaware of their existence. If one thread makes a blocking system call, the entire process (and all its threads) gets blocked. **Fast to create and switch, but not truly concurrent.**
    - **Kernel-level Threads:** Managed directly by the OS kernel. The kernel has a PCB for each thread. If one thread blocks, the kernel can schedule another thread from the same process. This allows for true concurrency on multi-core systems. **Slower to create and switch, but more robust and powerful.**
      Example: Windows and Linux use kernel-level threads.
16. What is process scheduling? Types of scheduling algorithms.
    Process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process from the ready queue. The goal is to maximize CPU utilization and minimize waiting time.
    Types of scheduling algorithms:
    - **First-Come, First-Served (FCFS)**
    - **Shortest Job First (SJF)**
    - **Priority Scheduling**
    - **Round Robin (RR)**
    - **Multilevel Queue Scheduling**
    - **Multilevel Feedback Queue Scheduling**
17. **Difference between preemptive and non-preemptive scheduling.**
    - **Preemptive Scheduling:** The OS can interrupt a running process and move it to the ready state, even if it has not completed. This is useful for time-sharing systems where a process shouldn't hold the CPU for too long. **Example:** Round Robin, Priority Scheduling.
    - **Non-preemptive Scheduling:** A process keeps the CPU until it either completes its execution or voluntarily moves to the waiting state. Once the CPU is allocated to a process, it cannot be taken away. **Example:** FCFS, SJF.
18. **Explain FCFS, SJF, Priority Scheduling, Round Robin.**
    - **FCFS (First-Come, First-Served):** The simplest algorithm. The process that arrives first is executed first. It's non-preemptive. **Drawback:** Can lead to the "convoy effect" where a single long job holds up many small jobs.
    - **SJF (Shortest Job First):** The process with the smallest execution time is executed next. Can be preemptive or non-preemptive. **Advantage:** Gives the minimum average waiting time. **Drawback:** Hard to predict the next CPU burst duration.
    - **Priority Scheduling:** Each process is assigned a priority, and the CPU is allocated to the process with the highest priority. Can suffer from **starvation**, where low-priority processes may never get to run.

- **Round Robin (RR):** A preemptive algorithm where each process is given a small unit of CPU time, called a **time quantum**. Once the quantum expires, the process is preempted and put back at the end of the ready queue. **Good for time-sharing systems.**

19. **What is starvation and aging?**
    - **Starvation:** A situation where a process is perpetually denied access to a resource or the CPU, even though the resource is available. This often happens in priority-based scheduling where low-priority processes never get to run.
    - **Aging:** A technique used to prevent starvation. The priority of a process that has been waiting in the ready queue for a long time is gradually increased over time. Eventually, its priority becomes high enough to get scheduled.
20. **Difference between cooperative multitasking and preemptive multitasking.**
    - **Cooperative Multitasking:** The OS relies on processes to voluntarily give up control of the CPU. If a process doesn't yield, it can freeze the entire system. **Example:** Early versions of macOS and Windows.
    - **Preemptive Multitasking:** The OS can forcibly interrupt a running process and assign the CPU to another one. This makes the system more responsive and prevents a single rogue process from freezing the entire system. **Example:** All modern operating systems like Windows 10, macOS, and Linux.

---

3. Memory Management

21. What is virtual memory?
    Virtual memory is a memory management technique that allows a program to use more memory than is physically available in the system. The OS creates an illusion of a very large, contiguous memory space. It does this by using a portion of the hard drive (swap space) as an extension of RAM. Data is swapped between RAM and the hard drive as needed. This allows multiple large programs to run simultaneously.
22. **Difference between paging and segmentation.**
    - **Paging:** A memory management technique where the logical address space is divided into fixed-size blocks called **pages**. The physical memory is divided into fixed-size blocks of the same size called **frames**. It avoids external fragmentation.
    - **Segmentation:** A memory management technique where the logical address space is divided into variable-size segments, each corresponding to a logical unit (e.g., a function, an array). It's more user-friendly but can lead to external fragmentation.
23. What is page fault?
    A page fault is an event that occurs when a program tries to access a page of data or code that is currently not in physical memory (RAM). The OS detects this, and the process is temporarily suspended while the missing page is loaded from the hard drive into an available frame in RAM. The process is then resumed.
24. **Difference between logical and physical address.**

    - **Logical Address (Virtual Address):** An address generated by the CPU. It's a "virtual" address that refers to a location within the process's address space. It's independent of the physical memory location.
    - **Physical Address:** An actual address in the main memory (RAM). The Memory Management Unit (MMU) is responsible for translating logical addresses to physical addresses.
25. Explain Demand Paging.
    Demand paging is a virtual memory technique where pages are not loaded into memory until they are needed (on "demand"). This is the most common form of virtual memory. When a process starts, only the first few pages are loaded. Subsequent pages are only loaded when a page fault occurs. This reduces the amount of I/O required and allows for more processes to be in memory simultaneously.
26. What is Thrashing?
    Thrashing is a situation where the system spends more time paging (swapping pages between memory and disk) than it does executing application instructions. This occurs when the working set of processes (the set of pages they are actively using) is larger than the available physical memory. The system becomes very slow, and CPU utilization drops drastically.
27. Explain TLB (Translation Lookaside Buffer).
    The Translation Lookaside Buffer (TLB) is a small, fast hardware cache on the CPU that stores recent logical-to-physical address translations. When the CPU needs to access a memory location, it first checks the TLB. If the translation is in the TLB (TLB hit), it's very fast. If not (TLB miss), the CPU has to access the main memory page table, which is slower. The goal of the TLB is to speed up address translation.
28. **What is fragmentation (internal vs external)?**
    - **Fragmentation:** The inability of memory to be used effectively due to being broken into small, non-contiguous blocks.
    - **Internal Fragmentation:** Occurs when memory is allocated in fixed-size blocks, and the allocated memory is larger than what is requested. The unused portion within the block is wasted. **Example:** If memory is allocated in 4 KB pages, and a process only needs 3 KB, 1 KB is wasted.
    - **External Fragmentation:** Occurs when there's enough total free memory to satisfy a request, but the free memory is not contiguous. It's scattered in small chunks, making it unusable for larger requests.
29. What is swapping in OS?
    Swapping is a memory management scheme where a process is temporarily moved from main memory to a backing store (like a hard disk) and then brought back into main memory for continued execution. It's used to manage memory when the total size of all processes exceeds the physical memory. The process that is "swapped out" is moved to a disk, freeing up memory for another process.
30. **Explain memory allocation methods (First Fit, Best Fit, Worst Fit).**
    - **First Fit:** The OS scans the memory from the beginning and allocates the **first free block** it finds that is large enough to satisfy the request. It's simple and fast.
    - **Best Fit:** The OS searches the entire free list and allocates the **smallest free block** that is large enough. This minimizes internal fragmentation but can be

slow as it has to search the entire list. It also tends to leave many tiny, unusable holes.
    - **Worst Fit:** The OS allocates the **largest free block** available. The idea is to leave a large enough remainder to be useful for other requests. This can be slow and may result in a rapid fragmentation of the memory.

---

4. Synchronization & Concurrency

31. What is process synchronization?
    Process synchronization is the coordination of processes that share a common resource. It ensures that multiple processes or threads accessing a shared resource do so in a controlled manner, preventing race conditions and data inconsistencies. Example: A producer-consumer problem where one thread adds data to a buffer and another removes it.
32. What is race condition?
    A race condition is a situation where the final outcome of a program depends on the non-deterministic order of execution of multiple processes or threads. It happens when multiple threads try to access and modify the same shared data concurrently. Example: Two threads trying to increment a counter simultaneously. The final value may be incorrect if the operations aren't atomic.
33. What are critical sections?
    A critical section is a segment of code where a shared resource is accessed and modified. The key principle is that only one process or thread should be allowed to execute its critical section at a time. The rest of the processes should be excluded. The goal of synchronization is to provide mutual exclusion for critical sections.
34. Explain Peterson’s Solution.
    Peterson's Solution is a classic software-based solution to the critical section problem for two processes. It uses two shared variables: turn (to indicate whose turn it is) and flag (an array to indicate if a process wants to enter the critical section). It satisfies the three requirements of critical section solutions: mutual exclusion, progress, and bounded waiting.
35. What are semaphores? Types of semaphores.
    A semaphore is a signaling mechanism used for process synchronization. It's an integer variable that is accessed only through two atomic operations: wait() (or P()) and signal() (or V()).
    - **Binary Semaphore:** Can only have a value of 0 or 1. Used to implement mutual exclusion, like a mutex.
    - **Counting Semaphore:** Can have an integer value greater than or equal to 0. Used to control access to a resource that has multiple instances. The value of the semaphore indicates the number of available resources.
36. **Difference between binary semaphore and mutex.**
    - **Mutex (Mutual Exclusion):** A lock that ensures only one thread can access a shared resource at a time. The thread that acquires the lock is the only one that can release it. Think of it as a key to a room. Only the person with the key can enter and exit.

    - **Binary Semaphore:** A signaling mechanism. A thread can acquire a semaphore and another can release it. It's used for synchronization between threads. It's like a signal flag that can be raised (1) or lowered (0) by any thread.
37. What is monitor in OS?
    A monitor is a high-level synchronization construct that simplifies the implementation of mutual exclusion. It's an abstract data type that contains a shared resource and a set of procedures for accessing it. A key feature is that only one process can be active inside the monitor at any given time, ensuring mutual exclusion. Monitors use condition variables for more complex synchronization needs.
38. Explain Producer-Consumer problem.
    The Producer-Consumer problem is a classic synchronization problem. A producer process generates data and puts it into a shared buffer. A consumer process takes data from the buffer and uses it. The problem is to ensure that the producer doesn't try to add data to a full buffer and the consumer doesn't try to remove data from an empty buffer. This is solved using semaphores (or monitors) to signal when the buffer is full or empty.
39. Explain Reader-Writer problem.
    The Reader-Writer problem involves a shared resource that multiple processes need to access. Some processes (readers) only need to read the data, while others (writers) need to modify it. The challenge is to allow multiple readers to access the data simultaneously, but to ensure that a writer has exclusive access, meaning no other readers or writers can access the resource while a writer is active.
40. Explain Dining Philosophers problem.
    The Dining Philosophers problem is a classic synchronization problem that illustrates the potential for deadlock. Five philosophers are sitting at a round table, with a single fork between each pair. To eat, a philosopher needs to pick up both forks on their left and right. The problem is how to design a protocol that allows philosophers to eat without getting into a deadlock, where each philosopher holds one fork and waits for the other, forever.

---

5. Deadlocks

41. What is deadlock?
    A deadlock is a situation in a multi-process system where two or more processes are blocked forever, each waiting for a resource that is held by another process in the group. Example: Process A holds resource R1 and waits for R2. Process B holds R2 and waits for R1. Both are indefinitely blocked.
42. What are the necessary conditions for deadlock?
    Four conditions must exist simultaneously for a deadlock to occur:
    - **Mutual Exclusion:** At least one resource must be non-shareable. Only one process can use it at a time.
    - **Hold and Wait:** A process is holding at least one resource and is waiting to acquire additional resources held by other processes.
    - **No Preemption:** Resources cannot be forcibly taken away from a process. They must be released voluntarily.

- **Circular Wait:** A set of processes are in a circular chain, where each process is waiting for a resource held by the next process in the chain.

43. **Difference between deadlock prevention, avoidance, and detection.**
    - **Prevention:** Ensures that at least one of the four necessary conditions for deadlock never occurs. **Example:** For "hold and wait," we can require a process to request all its resources at once.
    - **Avoidance:** Requires some advance knowledge of the resource needs of processes. The system checks each resource request to ensure that granting it won't lead to an unsafe state (a state from which deadlock can't be avoided). **Example:** Banker's Algorithm.
    - **Detection:** Allows a deadlock to occur. The system periodically runs an algorithm to detect if a deadlock has occurred. If one is found, it's resolved by a recovery mechanism.
44. What is Banker's Algorithm?
    The Banker's Algorithm is a deadlock avoidance algorithm. It determines if a new resource request from a process can be granted without putting the system in an unsafe state. It's called "banker's" because it's analogous to a banker who doesn't lend money if it can't be repaid. The algorithm ensures there's a "safe sequence" of processes that can be completed.
45. **Difference between deadlock and starvation.**
    - **Deadlock:** A state where processes are **permanently blocked** and cannot make any progress. The system is at a standstill.
    - **Starvation:** A state where a process is **repeatedly denied** access to a resource, even though the resource becomes available. The process is not blocked permanently but keeps getting bypassed by other processes. Starvation can be temporary, while a deadlock is a permanent blockage.
46. What is livelock?
    A livelock is a situation where two or more processes are continuously changing their state in response to each other, but without making any useful progress. They are not blocked but are perpetually busy, constantly reacting to each other's actions, leading to a state where neither can proceed. It's similar to two people trying to pass each other in a narrow hallway, constantly stepping aside and back in sync.
47. How to recover from deadlock?
    Once a deadlock is detected, recovery methods include:
    - **Process Termination:**
        - Abort all deadlocked processes.
        - Abort one process at a time until the deadlock is resolved.
    - **Resource Preemption:**
        - Take a resource away from a process and give it to another. The preempted process is then rolled back to a safe state. This requires the ability to save and restore process state.
        - Selecting the victim is critical (e.g., aborting a low-priority or least-cost process).

---

6. File System & I/O

48. What is file system?
    A file system is a method and data structure used by the OS to manage and organize files and directories on a storage device (like a hard disk or SSD). It provides a logical, hierarchical view of the data, allowing users to easily create, delete, read, and write files. It manages access control, file metadata (like size and date created), and the physical location of the data.
49. **What are different file access methods (sequential, direct, indexed)?**
    - **Sequential Access:** Files are accessed in a linear, ordered manner, from beginning to end. **Example:** Reading a text file from start to finish.
    - **Direct Access (Relative Access):** Records can be accessed directly and in any order. Each record has a logical record number. The OS translates this logical number to a physical address. **Example:** Databases.
    - **Indexed Access:** A file has an index block that contains pointers to the data blocks. To access a record, the system first reads the index block and then uses the pointer to directly access the data block. **Example:** Large databases and file systems like NTFS.
50. **Difference between FAT32, NTFS, ext4.**
    - **FAT32 (File Allocation Table 32):** An older file system used by older versions of Windows and USB drives. **Limitations:** No security features, a 4 GB maximum file size, and a 2 TB maximum partition size.
    - **NTFS (New Technology File System):** A modern file system used by all recent versions of Windows. **Features:** Journaling, file and folder permissions (security), support for larger files and partitions, and data compression.
    - **ext4 (Fourth Extended Filesystem):** The standard file system for most Linux distributions. **Features:** Journaling, support for large files (up to 16 TB), and robust performance.
51. What is inode in Linux?
    An inode (index node) is a data structure in Linux/Unix file systems that stores information about a file or directory. It does not store the file's content or its name. The inode contains:
    - **Metadata:** File type (regular file, directory, etc.), permissions, owner, group.
    - **Timestamps:** Time of creation, last modification, and last access.
    - **Pointers to Data Blocks:** The physical disk addresses where the file's data is stored.
52. Explain directory structure in OS.
    A directory structure is the hierarchical organization of files on a storage device.
    - **Single-Level Directory:** All files are in a single directory. Simple, but hard to manage as the number of files grows.
    - **Two-Level Directory:** Each user has their own separate directory, but they are all at the same level.
    - **Tree-Structured Directory (most common):** Files are organized in a tree-like hierarchy. It allows for grouping related files and provides a logical structure. **Example:** `/home/user/documents/work.txt`.
53. What is journaling in file systems?
    Journaling is a feature in file systems that records changes to the file system's metadata in a circular log called a journal before the changes are actually made to

the main file system. In case of a system crash, the file system can use the journal to recover and restore a consistent state, preventing data corruption. Example: NTFS and ext4 use journaling.

54. Explain RAID levels (RAID 0, 1, 5, 10).
    RAID (Redundant Array of Independent Disks) is a technology that combines multiple physical disk drives into a single logical unit to improve performance or provide redundancy.
    - **RAID 0 (Striping):** Data is striped across multiple disks. Improves performance but provides no redundancy. If one drive fails, all data is lost.
    - **RAID 1 (Mirroring):** Data is written identically to two or more disks. Provides excellent data redundancy but uses 50% of the disk space.
    - **RAID 5 (Striping with Parity):** Data is striped across multiple disks, and a parity block is also distributed. Provides fault tolerance and good performance. Can tolerate a single disk failure.
    - **RAID 10 (RAID 1+0):** Combines striping and mirroring. Data is mirrored in pairs, and the mirrored pairs are then striped. Provides both high performance and excellent redundancy.
55. **What is buffer vs cache in OS?**
    - **Buffer:** A temporary storage area used to hold data during I/O operations. It's used to compensate for speed differences between the producer and consumer of data. **Example:** A keyboard buffer stores keystrokes before they are processed.
    - **Cache:** A smaller, faster memory used to store copies of frequently accessed data from a slower storage. Its purpose is to reduce the average time to access data. **Example:** The CPU cache stores frequently used instructions and data from RAM.
56. What is spooling in OS?
    Spooling (Simultaneous Peripheral Operations On-Line) is a technique where the OS uses a buffer (a spool) on a disk to store data for a device, such as a printer. This allows a process to send its output to the printer and then terminate immediately, without having to wait for the printer to become available. The spooler process handles the transfer from the disk to the printer in the background.

---

7. Advanced OS Concepts

57. **What is interrupt vs polling?**
    - **Interrupt:** A hardware signal sent by a device to the CPU to request its attention. The CPU pauses its current task, saves its state, and jumps to an **Interrupt Service Routine (ISR)** to handle the request. This is efficient as the CPU isn't wasting cycles checking for events.
    - **Polling:** The CPU repeatedly checks the status of a device to see if it needs service. This is a CPU-intensive process and is inefficient if the device is slow or has infrequent events.
58. **Difference between trap and interrupt.**

    - **Interrupt:** A **hardware-generated** signal (e.g., from an I/O device) that a needs attention. It's an asynchronous event, meaning it can happen at any time.
    - **Trap:** A **software-generated** interrupt, triggered by an instruction. It's a synchronous event, meaning it happens at a specific point in the execution of a program. Traps are often used to invoke a system call or to handle errors like division by zero.
59. What is DMA (Direct Memory Access)?
    Direct Memory Access (DMA) is a hardware feature that allows an I/O device to transfer data directly to and from main memory without the constant involvement of the CPU. This frees up the CPU to perform other tasks, greatly improving system performance for large data transfers. The CPU only gets involved at the beginning and end of the transfer.
60. What is real-time operating system (RTOS)?
    An RTOS is an operating system designed to serve real-time applications where the processing of data must be completed within a strict time constraint. A key metric for an RTOS is not high throughput, but timeliness and predictability. The correctness of the system depends not only on the logical output but also on the time at which the result is produced. Example: Air traffic control systems, anti-lock brakes.
61. **Difference between hard real-time and soft real-time systems.**
    - **Hard Real-time:** A system where meeting deadlines is **absolutely critical**. Missing a deadline is considered a system failure and can lead to catastrophic consequences. **Example:** Pacemakers, flight control systems.
    - **Soft Real-time:** A system where meeting deadlines is desirable but not mandatory. Missing a deadline can degrade performance but doesn't cause a system failure. **Example:** Multimedia streaming, online gaming.
62. **What is multitasking vs multiprocessing?**
    - **Multitasking:** The ability of a **single CPU** to execute multiple tasks concurrently by rapidly switching between them. It creates the illusion of parallel execution.
    - **Multiprocessing:** The ability of a system with **multiple CPUs or cores** to execute multiple processes simultaneously. This is true parallel execution.
63. Explain message passing vs shared memory.
    These are two methods for Inter-Process Communication (IPC).
    - **Message Passing:** Processes communicate by exchanging messages. It's simpler to implement, works well for distributed systems, and provides isolation. **Drawback:** Slower due to the overhead of system calls for message transfer.
    - **Shared Memory:** Processes share a region of memory. Once set up, communication is very fast as it's just a memory access. **Drawback:** Requires careful synchronization to prevent race conditions.
64. What is priority inversion?
    Priority inversion is a scheduling problem where a high-priority process is blocked and forced to wait for a low-priority process to release a resource. This can happen if the low-priority process holds a lock that the high-priority process needs. A well-known solution is Priority Inheritance, where the low-priority process temporarily inherits the higher priority of the process it is blocking.

65. What is kernel panic?
    A kernel panic is a fatal error from which the operating system cannot safely recover. It's an internal, unrecoverable error in the kernel. When this happens, the kernel typically stops all activity, displays an error message, and halts the system to prevent data corruption. Example: The infamous Blue Screen of Death in Windows is a type of kernel panic.
66. **Difference between cooperative and preemptive scheduling.**
    - **Cooperative Scheduling:** Processes voluntarily give up the CPU. If a process does not yield, it can lock up the entire system.
    - **Preemptive Scheduling:** The OS can forcibly take the CPU away from a process. This provides better responsiveness and stability. All modern OSs use preemptive scheduling.

---

8. Linux/Unix Specific

67. **Difference between Linux and Windows OS.**
    - **Kernel:** Linux uses a monolithic kernel, while Windows uses a hybrid kernel.
    - **Source Code:** Linux is open-source and free, while Windows is proprietary and commercial.
    - **User Interface:** Windows is primarily GUI-based. Linux offers a choice of GUIs (e.g., GNOME, KDE) but is also heavily reliant on the command line.
    - **File System:** Linux primarily uses ext4, while Windows uses NTFS.
    - **Security:** Due to its open-source nature, Linux is often considered more secure, as vulnerabilities are found and patched quickly by the community.
68. What are system calls in Linux?
    A system call is the interface between a user-space program and the Linux kernel. It allows a program to request services from the OS, such as creating a process, accessing a file, or managing memory. Example: read(), write(), fork(), execve().
69. **Explain fork() and exec() in Linux.**
    - **`fork()`:** A system call that creates a **new process (child process)** by duplicating the calling process (parent process). The child is an exact copy of the parent, including its memory and file descriptors, but it has a unique Process ID (PID).
    - **`exec()`:** A family of system calls that replaces the current process's address space with a new program. After `exec()`, the original program is gone, and the new program starts its execution. It's often called after `fork()` to run a different program in the new child process.
70. **Difference between thread vs process in Linux.**
    - **Process:** A heavyweight unit of execution. Each process has its own separate address space. Communication between processes requires IPC mechanisms. Created with `fork()`.
    - **Thread:** A lightweight unit of execution within a process. Threads share the same address space. Communication is fast via shared memory. Created with `pthread_create()`.
71. What is init process in Linux?
    The init process is the first process (PID 1) started by the Linux kernel during

boot-up. It's the "mother of all processes." All other processes are descendants of init. Its job is to manage the system's startup, run scripts to start other services and daemons, and ensure the system reaches a specific runlevel (or a system state). Modern Linux distributions have replaced init with systemd.

72. **Difference between ps, top, and htop commands.**
    - **ps (Process Status):** Displays a static snapshot of the currently running processes. It doesn't update in real time. **Example:** `ps aux`.
    - **top:** Provides a real-time, dynamic view of a system's running processes. It shows information like CPU usage, memory usage, and process details. It updates periodically.
    - **htop:** An enhanced, more user-friendly version of `top`. It offers a more visually appealing and interactive interface, with colors, horizontal scrolling, and easy-to-use controls for managing processes.
73. What is context switching in Linux?
    In Linux, a context switch is the process of saving the state of a running process or thread and restoring the state of another one. The Linux kernel's scheduler (e.g., CFS) performs context switches to give each process or thread a turn on the CPU, enabling multitasking.
74. **Difference between kill, killall, and pkill commands.**
    - **kill:** Kills a single process by its **Process ID (PID)**. **Example:** `kill 1234`.
    - **killall:** Kills all processes by their **name**. **Example:** `killall firefox`.
    - **pkill:** Kills processes by their **name** or other attributes (e.g., user, parent process). It's more flexible than `killall` as it can use patterns and regular expressions. **Example:** `pkill -u user1`.
75. What are runlevels in Linux?
    Runlevels are modes of operation in a Linux system that define what services are running. They are numbered from 0 to 6.
    - **Runlevel 0:** Halt (power off the system).
    - **Runlevel 1:** Single-user mode (for maintenance).
    - **Runlevel 2:** Multi-user mode without networking.
    - **Runlevel 3:** Multi-user mode with networking (CLI).
    - **Runlevel 5:** Multi-user mode with networking and a GUI.
    - Runlevel 6: Reboot.
      Modern systems using systemd have largely replaced runlevels with "targets."

---

9. Security in OS

76. What are different types of OS security?
    OS security involves protecting the system from unauthorized access and damage.
    - **Authentication:** Verifying the identity of a user (e.g., username/password).
    - **Authorization:** Granting a user specific permissions to access resources (e.g., read/write permissions on a file).
    - **Access Control:** Mechanisms that enforce authorization policies, such as ACLs and Role-Based Access Control.
    - **Security Auditing:** Monitoring system activities to detect and record security events.

    - **Firewalls:** Protecting the system from network attacks.
77. **What is authentication vs authorization in OS?**
    - **Authentication:** The process of verifying a user's identity. It answers the question, **"Are you who you say you are?" Example:** Entering a username and password.
    - **Authorization:** The process of granting or denying a user access to a resource based on their verified identity. It answers the question, **"What are you allowed to do?" Example:** A user is allowed to read a file but not write to it.
78. What is access control list (ACL)?
    An Access Control List (ACL) is a list of permissions associated with a file, folder, or other object. It specifies which users or groups have what kind of access (e.g., read, write, execute). When a user tries to access a resource, the OS checks the ACL to determine if the access is permitted.
79. What are system vulnerabilities in OS?
    System vulnerabilities are weaknesses in the OS software or design that can be exploited by an attacker to gain unauthorized access or cause a system to fail. Example:
    - **Buffer overflow:** Writing more data to a buffer than it can hold, overwriting adjacent memory.
    - **Privilege escalation:** A user with low-level privileges gains high-level administrator privileges.
    - **Race conditions:** An attacker exploits a timing issue in the OS.
    - **Unpatched software:** Known security flaws in software that have not been fixed.
80. What is buffer overflow attack?
    A buffer overflow attack is an exploit where a program writes data to a buffer in memory that is larger than the buffer's allocated size. The excess data "overflows" the buffer and overwrites adjacent memory, which may contain critical data or instructions. An attacker can use this to inject malicious code into the system and execute it with the privileges of the vulnerable program.

---

10. Miscellaneous & Tricky

81. **Difference between multiprocessor and multicore.**
    - **Multiprocessor:** A computer system with **multiple, distinct CPUs** on the same motherboard. Each CPU has its own set of caches and buses.
    - **Multicore:** A single CPU chip (package) that contains **multiple processing cores**. The cores share some resources, like a common L2/L3 cache and the same bus to the main memory. Most modern computers use multicore processors.
82. What is time-sharing OS?
    A time-sharing OS is a type of multitasking OS that allows many users to share a single computer simultaneously. The CPU is rapidly switched between users' processes, giving each one a small slice of time (time quantum), creating the illusion that each user has exclusive access to the system. This allows for interactive use.

83. What is cooperative scheduling?
    Cooperative scheduling is a scheduling approach where a running process voluntarily gives up control of the CPU. If the process does not yield, it can monopolize the CPU, causing the entire system to appear frozen. It's not used in modern general-purpose OSs because of its lack of robustness.
84. **What is demand paging vs prepaging?**
    - **Demand Paging:** Pages are loaded into memory only when they are explicitly referenced (on demand). This is the most common virtual memory implementation.
    - **Prepaging:** The OS tries to predict which pages a process will need in the near future and loads them into memory *before* they are referenced. This can reduce page faults but can also lead to wasted I/O if the predictions are wrong.
85. **Difference between latency and throughput.**
    - **Latency:** The **time delay** between a request and a response. It's the time it takes to complete a single task. **Example:** The time it takes for a web page to load.
    - **Throughput:** The **rate** at which work is completed. It's the number of tasks completed per unit of time. **Example:** The number of HTTP requests a server can handle per second.
86. What is bootloader?
    A bootloader is a small program that is loaded by the computer's firmware (BIOS/UEFI) during the boot process. Its primary job is to load the operating system kernel into memory and start its execution. Example: GRUB (GRand Unified Bootloader) is a common bootloader for Linux.
87. What is swap space in Linux?
    Swap space is a dedicated area on a hard disk or SSD that is used by the OS as an extension of physical memory (RAM). When the amount of physical memory is low, the OS can move inactive pages of memory from RAM to swap space, freeing up RAM for active processes. It's part of the virtual memory system.
88. **What is hypervisor and virtualization in OS?**
    - **Virtualization:** A technology that creates a virtual version of a resource, such as an OS, a server, or a storage device. It allows a single physical machine to host multiple virtual machines (VMs).
    - **Hypervisor:** A layer of software that sits between the hardware and the VMs. Its job is to manage and run the VMs, allocating hardware resources to each one. It's the core of virtualization. **Example:** VMware, VirtualBox, Hyper-V.
89. **Difference between type 1 and type 2 hypervisors.**
    - **Type 1 Hypervisor (Bare Metal):** Runs directly on the host machine's hardware. It doesn't rely on an underlying OS. It's more efficient, secure, and used in enterprise environments. **Example:** VMware ESXi, Microsoft Hyper-V.
    - **Type 2 Hypervisor (Hosted):** Runs as an application on top of an existing OS. It's less efficient because it has to go through the host OS to access hardware. It's popular for desktop use. **Example:** VirtualBox, VMware Workstation.
90. **What is containerization (Docker vs VM)?**

- **Virtual Machine (VM):** A VM includes a full copy of the OS, applications, and their dependencies. It's large, slow to start, but provides excellent isolation. It virtualizes the entire hardware stack.
- **Containerization:** A technology that packages an application with all its dependencies into a single, isolated unit called a container. Containers share the host OS kernel. They are lightweight, fast to start, and use less resources. **Example:** Docker.

---

11. Performance & Optimization

91. **What is CPU burst and I/O burst?**
    - **CPU Burst:** A period of time when a process uses the CPU to perform calculations.
    - I/O Burst: A period of time when a process performs an I/O operation (e.g., reading from a disk).
      The performance of a process depends on the alternation of these two bursts. CPU-bound processes have long CPU bursts, while I/O-bound processes have short CPU bursts and long I/O bursts.
92. **What is turnaround time, waiting time, response time in scheduling?**
    - **Turnaround Time:** The total time from when a process is submitted to when it completes. It includes waiting time and execution time.
    - **Waiting Time:** The total time a process spends waiting in the ready queue for the CPU.
    - **Response Time:** The time from when a process is submitted to the first time it gets the CPU.
93. What is multi-level queue scheduling?
    Multi-level queue scheduling partitions the ready queue into multiple separate queues. Each queue may have its own scheduling algorithm. Processes are permanently assigned to a queue based on their properties (e.g., system processes, interactive processes, batch processes). For example, a system might have a high-priority queue for interactive jobs using Round Robin and a low-priority queue for batch jobs using FCFS.
94. What is multi-level feedback queue scheduling?
    This is an extension of the multi-level queue. A process can move between different queues. It's a more dynamic approach. A new process might start in the highest-priority queue. If it doesn't finish within a certain time (e.g., its time quantum expires), it is moved to a lower-priority queue. This prevents starvation and gives priority to interactive, short-burst processes.
95. **Difference between CPU-bound and I/O-bound processes.**
    - **CPU-bound Process:** A process that spends most of its time executing instructions on the CPU and performs little I/O. **Example:** A program that calculates a very large number, or a scientific simulation.
    - **I/O-bound Process:** A process that spends most of its time waiting for I/O operations to complete. It has short CPU bursts. **Example:** A text editor, a web browser.

96. What is load balancing in OS?
    Load balancing is the process of distributing computing workloads evenly across multiple resources, such as CPUs, servers, or networks. The goal is to maximize throughput, minimize response time, and avoid overloading any single resource.
97. How does OS handle I/O management?
    The OS handles I/O management through:
    - **Device Drivers:** Software components that provide an interface between the OS and the hardware device.
    - **I/O Scheduling:** Optimizing the order of I/O requests to maximize efficiency.
    - **Buffering and Caching:** Using temporary memory to hold data for I/O operations.
    - **Spooling:** Using a disk as a buffer to handle I/O to slower devices.
98. What is memory-mapped I/O?
    Memory-mapped I/O is a method of interacting with hardware devices where the device's control registers and data buffers are mapped to a section of the CPU's address space. The CPU can then access the device simply by performing standard memory read and write operations, eliminating the need for special I/O instructions.
99. What is NUMA (Non-Uniform Memory Access)?
    Non-Uniform Memory Access (NUMA) is a computer memory design used in multiprocessor systems where the access time to memory depends on the memory's location relative to the processor. A processor can access its local memory faster than it can access memory that is physically located with another processor. This is a key consideration for performance optimization on multi-socket servers.
100. Explain differences between distributed OS and network OS.
    * Distributed OS (DOS): Manages a collection of independent computers and makes them appear to the user as a single, unified system. The OS hides the underlying network details. It provides a single system image.
    * Network OS (NOS): Runs on each computer in a network and provides services like file sharing and network security. Each computer retains its own independent OS and a local file system. The user is aware that they are on a network. Example: Windows Server, Red Hat Enterprise Linux.