1. Explain the primary functions of an operating system and their importance.

The operating system (OS) is a software layer that manages hardware resources and provides services for applications.A program that acts as an intermediary between a user of a computer and the computer hardware. Its primary functions include:

- **Process Management:** Handles process creation, scheduling, and termination.
    - *Example:* The OS switches between running a text editor and a web browser.
- **Memory Management:** Allocates and deallocates memory for processes.
    - *Example:* When opening multiple applications, the OS ensures they don't overwrite each other's memory.
- **File System Management:** Controls file access, storage, and permissions.
    - *Example:* Reading/writing files in directories.
- **Device Management:** Manages input/output devices like keyboards, mice, and printers.
- **Security & Access Control:** Protects data and system integrity through authentication and permissions.

---

2. What is the kernel?

The **kernel** is the core component of an OS that runs in privileged mode and interacts directly with the hardware. It manages system resources such as CPU, memory, and device I/O.

- *Example:* When an application requests to read a file, the kernel translates that request into disk operations.

---

3. Explain what is the kernel mode of the CPU. Why is it necessary?

The **kernel mode** is a privileged CPU mode where the OS can directly access hardware and execute critical instructions.

- **Necessity:**
    - Ensures security by preventing user applications from directly accessing hardware.
    - Protects system stability by restricting direct memory access.
    - Allows the OS to manage multitasking and interrupts.
- *Example:* Writing to a hardware register (like controlling memory access) can only be done in kernel mode.

---

4. What is a system call? How does it differ from a regular function call?

A **system call** is an interface that allows user programs to request OS services, such as file operations and process management.

- **Differences from function calls:**
  - A function call is executed within user space, while a system call requires a mode switch to kernel space.
  - System calls involve additional overhead due to context switching.
- *Example:*
  - **Regular function call:** `printf("Hello")` writes to the screen.
  - **System call:** `write(fd, "Hello", 5)` makes a system call to write to a file.

---

5. What is a shell? Why is it important?

A **shell** is a command-line interface that allows users to interact with the OS by executing commands.

- **Importance:**
  - Enables automation through scripting.
  - Provides a way to manage files, processes, and system resources.
- *Example:*
  - Running `ls` lists files in a directory.
  - Running `bash script.sh` executes a script.

---

6. Differentiate between a process and a program, providing examples.

- A **program** is a set of instructions stored on disk.
- A **process** is a running instance of a program.
- *Example:*
  - `firefox` (program) is stored on disk.
  - When launched, `firefox` becomes a **process** with a unique Process ID (PID).

---

7. Define multiprogramming and time-sharing systems. How do they improve system performance?

- **Multiprogramming:** The OS keeps multiple programs in memory and switches between them to keep the CPU busy.
- **Time-sharing:** Extends multiprogramming by allowing rapid switching between tasks for interactive user experience.

- **Performance Benefits:**
    - Maximizes CPU utilization.
    - Reduces idle time.
    - Improves responsiveness.
- *Example:* Running a text editor, a browser, and a media player simultaneously.

---

8. What is concurrency? How can a single CPU achieve concurrency?

Concurrency means multiple tasks appear to execute at the same time.

- A single CPU achieves this through **context switching**, where it rapidly switches between tasks.
- *Example:* Downloading a file while browsing the internet.

---

9. Compare preemptive and non-preemptive scheduling. Provide advantages and disadvantages of each.

- **Preemptive:** The OS can interrupt a running process.
    - *Advantage:* Fairness between tasks.
    - *Disadvantage:* Context switching overhead.
- **Non-preemptive:** A process runs until it voluntarily releases the CPU.
    - *Advantage:* Simplicity and efficiency.
    - *Disadvantage:* Risk of starvation (longer jobs may block others).
- *Example:*
    - **Preemptive:** Round-Robin scheduling in modern OS.
    - **Non-preemptive:** First-Come, First-Served (FCFS) scheduling.

---

10. Define the role of a CPU scheduler. How does it influence performance and responsiveness?

A **CPU scheduler** selects which process gets CPU time based on scheduling algorithms.

- **Influence on performance:**
    - Reduces wait times.
    - Prevents CPU starvation.
    - Ensures fairness.
- *Example:* In an OS with multiple users, the scheduler ensures all users get CPU access fairly.

---

11. What is a time quantum in CPU scheduling? How does it affect performance?

A **time quantum** is the fixed time a process runs before being preempted.

- **Effects on performance:**
  - Short quantum → better responsiveness but high overhead.
  - Long quantum → efficient but slower response times.
- *Example:*
  - Quantum = 10ms: smooth multitasking.
  - Quantum = 1s: noticeable lag in responsiveness.

---

12. Describe a context switch. What steps are involved?

A **context switch** is when the CPU saves the state of a running process and loads another.

- **Steps:**
  1. Save process state (registers, stack, program counter).
  2. Update PCB.
  3. Load new process state.
  4. Resume execution.
- *Example:* Switching from a media player to a browser.

---

13. List and explain process states.

1. **New:** Process is created.
2. **Ready:** Waiting for CPU.
3. **Running:** Currently executing.
4. **Blocked:** Waiting for I/O.
5. **Terminated:** Process finished execution.

---

14. What is a Process Control Block (PCB)?

A **PCB** stores process information:

- Process ID (PID).
- Process state.
- CPU registers.
- Memory information.
- I/O details.
  Example: When resuming a paused program, the OS retrieves its PCB.

---

15. Define a logical memory address. How does it differ from a physical address?

- **Logical Address:** Used by programs (virtual memory).
- **Physical Address:** Actual memory location.
- **Purpose:** Allows memory protection and relocation.
- *Example:* Virtual memory allows running large programs without needing equivalent RAM.

---

16. What is a page fault? How is it handled?

A **page fault** occurs when a required page isn't in RAM.
Handling:

1. Suspend process.
2. Load page from disk.
3. Update page table.
4. Resume process.

---

17. Compare threads and processes.

- **Threads** share memory, while **processes** have separate memory.
- **Threads** are faster for communication.
- *Example:* A browser uses threads for tabs.

---

18. What is a deadlock? Provide an example.

A **deadlock** occurs when processes hold resources while waiting for each other indefinitely.

- *Example:* Process A locks a printer, Process B locks a scanner, both waiting for the other.

---

19. Define a race condition. Provide an example.

A **race condition** occurs when multiple threads modify shared data unpredictably.

- *Example:* Two threads incrementing a counter simultaneously without synchronization.

---

20. What is starvation? How can it be prevented?

Starvation happens when a process never gets CPU time due to priority scheduling.

- **Prevention:** Aging (gradually increasing priority of waiting processes).

---

21. Explain active waiting. Provide an example.

Active waiting is when a process continuously checks a condition instead of sleeping.

- *Example:* A loop checking for user input without pausing.