1. What happens step by step when you type a command in bash (e.g., ls) until you see the output?

- When I type a command in Bash, this is the sequence:

1. Input: You type ls and press Enter.
2. Parsing: The shell (bash) parses your command line, splitting it into command + arguments.
3. PATH Search: Bash looks for the binary/executable by searching through directories in the $PATH variable (/bin, /usr/bin, etc.).
4. Fork: Bash creates a child process using the fork () system call.
5. Exec: The child process replaces its memory with the ls program using execve().
6. System calls: The program makes system calls to the kernel (e.g., open(), readdir(), write ()) to list directory contents.
7. Output: The results are written to stdout (usually your terminal screen).
8. Exit status: The program ends, returns an exit code to the shell ($?), and bash waits for the next command.

- Example: When running ls, the kernel reads the directory contents and writes them back to your terminal.

2. Explain the types of processes in Linux: daemon, zombie, orphan. How can you detect them?

- Daemon Process

  It's a background process that's not tied to a terminal, Often started at boot and runs continuously, Detected using: ps -ef | grep daemon.

- Zombie Process

  A process that has finished execution, but its parent hasn't collected its exit status files so it still has an entry in the process table with state z, the kernel does a checkup and deletes it a while after, maybe after the parent process finishes execution. Detect with: ps aux | grep Z.

- Orphan Process

A process whose parent has exited while it's still running, the init process that has PID 1, probably system, adopts it, detected with: ps -ef –forest, you'll see the parent as systemd.

3. Why do we need Inter-Process Communication (IPC)? List some IPC mechanisms and real-life examples.

- As processes are isolated in Linux, so we need them to collaborate (share data, synchronize actions, send signals), they need communication channels. It allows multiple programs to work together efficiently.
- IPC Mechanisms in Linux:

1. Pipes / Named Pipes (FIFO)
    - Unidirectional communication.
    - Example: ls | grep ".txt" (stdout of ls → stdin of grep).
2. Signals
    - Used to notify processes of events.
    - Example: kill -9 <PID> sends SIGKILL.
3. Message Queues
    - Processes send and receive messages in a queue.
    - Example: communication between system services.
4. Shared Memory
    - Fastest method: processes share a memory segment.
    - Example: databases like PostgreSQL use it for caching.
5. Sockets
    - Allow communication between processes, even across machines.
    - Example: a web browser communicates with a web server via TCP sockets.