1. Process Concept

A **process** is a running instance of a program. It includes the program code, current activity represented by the program counter, the contents of the processor's registers, the process stack containing temporary data, and the data section containing global variables.

- **Components of a Process:**
  - **Program Counter (PC):** Points to the next instruction to be executed.
  - **Stack:** Holds temporary data like function parameters, return addresses, and local variables.
  - **Data Section:** Contains global variables.
  - **Heap:** A dynamically allocated memory for variables.
- **Process Control Block (PCB):** A data structure in the operating system that contains information about a process, such as:
  - **Process ID (PID):** Unique identifier for each process.
  - **Process State:** Tracks if the process is new, running, waiting, etc.
  - **Program Counter:** Stores the next instruction to execute.
  - **CPU Registers:** Stores contents of all process registers.
  - **Memory Management Info:** Information about memory allocated to the process.
  - **I/O Status:** I/O devices allocated to the process.
- **Process States:** These are the different stages a process can be in during its lifecycle:
  - **New:** The process is being created.
  - **Ready:** The process is waiting to be assigned to the CPU.
  - **Running:** Instructions are being executed.
  - **Waiting (Blocked):** The process is waiting for some event (e.g., I/O completion).
  - **Terminated:** The process has completed execution.

2. Process Scheduling

Scheduling is essential to ensure that the CPU is used efficiently and that users can interact with the system.

- **Schedulers:** Different types of schedulers handle different tasks.
  - **Long-Term Scheduler:** (Job Scheduler) Decides which processes should be brought into the ready queue. It controls the degree of multiprogramming (number of processes in memory).
  - **Short-Term Scheduler:** (CPU Scheduler) Selects a process from the ready queue and assigns it to the CPU for execution.
  - **Medium-Term Scheduler:** Swaps out processes from memory to reduce the load (context switching) and then reintroduces them when necessary.
- **Scheduling Algorithms:**
  - **First-Come, First-Served (FCFS):** Processes are executed in the order they arrive. It is simple but can lead to poor performance due to the **convoy effect** (when short processes wait for long ones).

    - **Shortest Job Next (SJN) / Shortest Job First (SJF):** The process with the shortest burst time (execution time) is selected first.
        - **Preemptive:** If a new process arrives with a shorter burst time than the current one, it preempts the running process.
        - **Non-Preemptive:** Once the process starts executing, it cannot be stopped until it completes.
    - **Priority Scheduling:** Each process is assigned a priority, and the CPU is allocated to the process with the highest priority.
        - **Preemptive:** A process with a higher priority can preempt a currently running process.
        - **Non-Preemptive:** Once a process starts, it runs until completion regardless of incoming higher priority processes.
    - **Round Robin (RR):** Each process is assigned a small fixed time quantum (time slice). After this time expires, the process is moved to the back of the ready queue.
    - **Multilevel Queue Scheduling:** Processes are grouped into different queues based on properties (e.g., system processes, interactive processes), and each queue has its own scheduling algorithm.
    - **Multilevel Feedback Queue:** Similar to multilevel queue scheduling but allows processes to move between queues based on their behavior.

3. Operations on Processes

Processes can interact in various ways:

- **Process Creation:**
    - **Parent and Child Processes:** A parent process creates a child process using system calls like `fork()` or `exec()`. The parent and child processes can share resources or be independent.
    - **System Call for Creation:**
        - **`fork()`:** Duplicates the calling process. The new process (child) is an exact copy of the calling (parent) process.
        - **`exec()`:** Replaces the current process memory with a new program.
        - **`wait()`:** The parent process can wait for its child to finish using the `wait()` system call.
- **Process Termination:**
    - A process can terminate when:
        - It finishes executing its final statement.
        - The process encounters an error and is terminated by the system.
        - A parent process can terminate a child process (e.g., by using the `kill()` system call).
- **Orphan and Zombie Processes:**
    - **Orphan Process:** A child process whose parent has terminated. The operating system adopts these processes.

    - **Zombie Process:** A terminated process that still has an entry in the process table.

4. Interprocess Communication (IPC)

IPC allows processes to exchange data and synchronize their execution. It can be achieved using two main methods:

- **Shared Memory:** Processes share a region of memory.
- **Message Passing:** Processes send messages to each other.

IPC ensures that processes can work together efficiently and is crucial for process synchronization.

5. IPC in Shared-Memory Systems

In shared-memory IPC, multiple processes have access to a shared memory segment. They can read and write to this region as needed.

- **Shared Memory System:**
    - The OS maps a section of physical memory into the address space of one or more processes.
    - **Concurrency Issues:** Shared-memory systems must address problems like race conditions, where multiple processes attempt to modify the same memory simultaneously.
- **Synchronization Tools:**
    - **Mutex (Mutual Exclusion):** A lock mechanism that ensures only one process can access a critical section of code at a time.
    - **Semaphores:** A signaling mechanism used to manage access to shared resources. There are two types:
        - **Binary Semaphore:** Acts as a simple lock (0 or 1).
        - **Counting Semaphore:** Keeps track of the number of available resources.
    - **Monitors:** A higher-level synchronization construct that combines mutexes and condition variables for easier management of concurrent processes.

6. IPC in Message-Passing Systems

Message passing allows processes to communicate without shared memory. Instead, they send and receive messages.

- **Direct Communication:** Processes must know each other explicitly (e.g., `send(processA, message)`).
- **Indirect Communication:** Processes use **mailboxes** or **message queues** to send and receive messages without knowing each other's identity.
- **Types of Message Passing:**

    - **Synchronous Message Passing:** The sender waits for the receiver to acknowledge the message.
    - **Asynchronous Message Passing:** The sender sends the message and continues execution without waiting for an acknowledgment.
- **Message Format:** Messages generally contain:
    - **Header:** Contains information like message size, type, and sender ID.
    - **Body:** Contains the actual data being communicated.

7. Examples of IPC Systems

Here are some widely used IPC mechanisms in modern operating systems:

- **POSIX Shared Memory:** A standard for creating and managing shared memory between processes. Functions like `shm_open()` and `mmap()` are used.
- **Message Queues (SysV IPC):** Allows processes to communicate by sending and receiving messages in queues. Functions like `msgsnd()` and `msgrcv()` are used.
- **Pipes:**
    - **Unnamed Pipes:** Typically used for communication between related processes. Data flows in one direction.
    - **Named Pipes (FIFOs):** Can be used for communication between unrelated processes. Data can flow in both directions.

8. Communication in Client-Server Systems

In the client-server model, a **server** provides services or resources, and a **client** requests them.

- **Sockets:** A socket is one endpoint of a two-way communication link between two programs running on the network. There are different types of sockets:
    - **Stream Sockets (TCP):** Provides reliable, connection-oriented communication.
    - **Datagram Sockets (UDP):** Provides connectionless, unreliable communication.
- **Remote Procedure Call (RPC):** Allows a client to invoke procedures on a remote server as if they were local procedures. RPC handles the communication details, providing an easy-to-use interface.
- **Client-Server Communication Types:**
    - **Connection-Oriented (e.g., TCP):** Reliable communication where the connection is established before data is transferred.
    - **Connectionless (e.g., UDP):** Unreliable communication where data is sent without establishing a connection.
-