**Module 2**

3. Know about the following topics and explore them (Write a note on your

understandings)

Areas for exploration,

● Child process - fork()

● Handling common signals

● Exploring different Kernel crashes

● Time complexity

● Locking mechanism - mutex/spinlock

**1. Child Process - fork()**

The fork() system call in Unix/Linux creates a **new child process** by duplicating the calling (parent) process. Both processes continue execution from the point of the fork but have separate address spaces.

* **Returns:**
  + In parent: PID of the child.
  + In child: 0.
  + On failure: -1.
* **Use cases:**
  + Creating new processes for concurrency.
  + Executing different tasks by child and parent.
  + Commonly combined with exec() to run a new program in the child.
* **Important:**  
  After fork(), parent and child have identical memory but separate copies due to **copy-on-write**.

**2. Handling Common Signals**

Signals are asynchronous notifications sent to processes to notify them of events like interrupts, exceptions, or termination requests.

* **Common signals:**
  + SIGINT (Ctrl+C) – Interrupt.
  + SIGTERM – Termination request.
  + SIGSEGV – Segmentation fault.
  + SIGKILL – Force kill (cannot be caught).
  + SIGALRM – Timer alarm.
* **Handling signals:**
  + Using signal() or sigaction() to define custom handlers.
  + Can choose to ignore, catch, or default action.
  + Important for graceful termination, cleanup, or recovery.

**3. Exploring Different Kernel Crashes**

Kernel crashes refer to failures in the operating system kernel, often resulting in **system panic** or **blue screen**.

* **Common causes:**
  + Null pointer dereference.
  + Deadlocks or resource starvation.
  + Faulty device drivers.
  + Hardware faults.
  + Stack overflows or memory corruption.
* **Kernel panic:**  
  OS halts to avoid corruption or undefined behavior, logs error info (kernel stack trace, error codes).
* **Debugging:**
  + Use tools like kdump, crash, or kernel logs (dmesg).
  + Reproduce issues with stress tests or fault injection.

**4. Time Complexity**

Time complexity is a measure of how the execution time of an algorithm scales with input size.

* **Big O notation:** Describes the upper bound of growth.
  + Examples: O(1), O(log n), O(n), O(n log n), O(n²), etc.
* **Why important:**
  + Predict performance.
  + Compare algorithms.
  + Optimize code for large inputs.
* **Examples:**
  + Searching in a sorted array by binary search: O(log n).
  + Bubble sort: O(n²).
  + Hash table lookup: O(1) average.

**5. Locking Mechanism - Mutex / Spinlock**

Locks are used in concurrent programming to **protect shared resources** and prevent race conditions.

* **Mutex (Mutual Exclusion):**
  + A lock that puts the thread to sleep if the lock is not available.
  + Suitable for longer wait times.
  + Prevents busy waiting.
* **Spinlock:**
  + Thread loops (spins) checking for lock availability without sleeping.
  + Useful when wait times are expected to be very short.
  + Can waste CPU cycles if held long.
* **Choosing between them:**
  + Use mutex for blocking locks.
  + Use spinlock in low-latency, high-frequency locking where sleep overhead is costly.
* **Implementation details:**
  + Often implemented using atomic instructions.
  + Critical for avoiding deadlocks, starvation.