**Moulik Tammana**

**Advance C Programming Module – 2**

**Areas for exploration: Child process - fork(), Handing common signals, Exploring different Kernel crashes, Time complexity and Locking mechanism - mutex/spinlock**

**1.Child Process Creation with fork ():**

* **The fork () system calls in Unix/Linux spawns a new child process.**
* **Initially, parent and child share the same code and memory, but execute independently.**
* **Copy-On-Write (COW) ensures that memory is only duplicated if either process modifies it, minimizing unnecessary copying.**
* **Proper management is needed to prevent zombie (terminated but unreaped) and orphan (parent exited) processes. F**
* **Fork () return values:**

**-1: Error occurred.**

**0: Running in the child process.**

**>0: Running in the parent (value is the child’s PID).**

* **To avoid zombie processes, the parent should use wait() or waitpid() after forking.**

**2. Handling Common Signals:**

* **Signals are asynchronous software interrupts used for process communication and control.**
* **Frequently encountered signals:**

1. **SIGINT (2): Sent by Ctrl+C to interrupt a process.**
2. **SIGTERM (15): Requests graceful termination.**
3. **SIGKILL (9): Forces immediate termination (cannot be caught or ignored).**
4. **SIGSEGV (11): Indicates invalid memory access (segmentation fault).**
5. **SIGCHLD (17): Notifies parent when a child process terminates.**
6. **SIGUSR1, SIGUSR2: Reserved for user-defined purposes.**

* **Register custom handlers with signal() or sigaction() to manage cleanup and graceful shutdowns.**
* **Long-running or resource-heavy programs should always handle SIGINT and SIGTERM for proper resource management.**

**3. Kernel Crash Types**

|  |  |  |
| --- | --- | --- |
| **Crash Type** | **Typical Cause** | **System Response** |
| **Kernel Oops** | **Minor kernel error, e.g., invalid access** | **Error is logged; system may continue running** |
| **Kernel Panic** | **Severe kernel fault, e.g., null pointer** | **System halts or restarts** |
| **Seg fault** | **Invalid memory access in user space** | **Process is terminated** |
| **Deadlock** | **Threads waiting indefinitely for locks** | **System may freeze or slow down** |
| **Hardware Error** | **Faulty RAM, overheating, power issues** | **Kernel reports an error** |
| **Watchdog Timeout** | **Missed system heartbeat** | **Automatic system reboot** |
| **Invalid Opcode** | **CPU encounters unknown instruction** | **May cause oops or panic** |

**4. Understanding Time Complexity**

* **Time complexity describes how the runtime of an algorithm scales with input size n*n*.**
* **Common complexity classes:**

1. **O (1): Constant time (e.g., accessing an array element)**
2. **O (log n): Logarithmic (e.g., binary search)**
3. **O (n): Linear (e.g., scanning a list)**
4. **O (n log n): Log-linear (e.g., merge sort)**
5. **O (n^2): Quadratic (e.g., bubble sort)**
6. **O (2^n): Exponential (e.g., naive Fibonacci)**

* **Big O represents the worst-case scenario, Omega the best, and Theta the average.**
* **This analysis ignores constants and lower-order terms, focusing on growth rates.**
* **Critical for designing efficient systems, especially in embedded, real-time, or large-scale environments.**

**5. Mutexes vs. Spinlocks**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Mutex** | **Spinlock** |
| **How it works** | **Thread sleeps if lock isn’t available** | **Thread loops until lock is free** |
| **Best for** | **Locks held for longer durations** | **Very short critical sections** |
| **Blocking** | **Yes, may trigger context switch** | **No, but uses CPU cycles** |
| **Pre-emption** | **Safe** | **Not safe; may need to disable pre-emption** |
| **Deadlock Risk** | **Yes, especially with circular waits** | **Yes, if not used carefully** |
| **Overhead** | **Higher for quick operations** | **Lower for short ops, but can waste CPU** |
| **Recursive Use** | **Supported with recursive mutexes** | **Not typically supported** |