embedKar i

Linux Application Development Training

Linux Application Development Training Will Power

This intensive training program bridges the gap between high-level application development and low-level Linux internals. Dive deep into kernel mechanisms, process architectures, and high-performance multi-threading. You will gain the exact skills required to build, debug, and optimize robust software on enterprise Linux infrastructure.

Rebar

Linux Overview Working with GNU Tools

- Kernel vs User Space: Master the fundamental separation of privileges. Learn how user applications securely interact with the core Linux kernel.
- Application-Level Debugging: Utilize hardware exceptions and core dumps to debug user-space applications.

File System Process Management

- **Process Anatomy**: Inspect the task_struct, memory layout, and environment blocks.
- **Control Flows**: Fork, execute, and monitor processes safely using fork(), exec(), and wait().

Synchronization Techniques

- Kernel Primitives: Deploy Mutexes, Semaphores, and Spinlocks appropriately.
- Lock-Free Concepts: Explore atomic operations and condition variables for fast signaling.

Inter-Process Communication (IPC)

- Data Pipelines: Stream high-throughput data using anonymous and named pipes (FIFOs).
- System V & POSIX API: Utilize shared memory, message queues, and memory-mapped files (mmap).
- Network IPC: Build local and remote communication pipelines using Unix Domain Sockets.

Thread-Level Programming

- **POSIX Threads**: Spawn, detach, and terminate execution tracks via the pthread library.
- **Thread Safety**: Author re-entrant code and manage thread-specific data (TSD).
- **Performance Tuning**: Implement custom thread pools and manage CPU core affinity.
-

Access Period: 12 Months

C Refresher with Intel Architecture

- Intel Architecture Blueprint: Demystify the CPU basics and general-purpose registers
- Call Stack & Frame Pointers: Trace execution paths through memory stack frames to visualize how function calls operate.

Why Linux Application development ?

- **Write High-Performance Code**: Master memory management, concurrency primitives, and defensive coding techniques.
- **Bridge Hardware & Software**: Learn how C code translates directly to registers, assembly, and cache lines.
- **Understand System Internals**: Demystify page tables, virtual memory, and hardware-enforced protection zones (Ring 0 vs. Ring 3).
- **Eliminate Guesswork Debugging**: Use system signals, core dumps, and kernel telemetry to isolate root causes instantly.