Embitude

Let's have fun with Embedded

Project-Oriented Embedded Linux Device Drivers

Description

This Project-Oriented Embedded Linux Device Drivers Training is designed to help engineers gain deep, practical expertise in Linux Kernel Internals and Device Driver Development for embedded systems. The program starts from the fundamentals — Linux kernel structure, modules, and character drivers — and progresses into advanced kernel concepts such as process management, synchronization, interrupt handling, and deferred work mechanisms. Each topic is paired with hands-on exercises, real-world problem statements, and industry-grade project challenges to ensure participants develop true, job-ready skills in Linux driver development

Key Highlights

- **Project-Oriented Learning**: Every concept reinforced with guided, practical driver development exercises.
- **Industry-Grade Assignments**: Solve realistic driver implementation problems to simulate actual embedded development scenarios.
- **Weekly Live Meetups**: Instructor-led sessions for concept reinforcement, assignment review, and peer discussions.
- **Lifetime Access**: Access all recorded sessions, reference materials, and updates at any time.
- **Capstone Project**: Apply all learned concepts in a comprehensive, industry-level driver integration project.
- **Mentor Support**: Learn directly from industry experts with deep hands-on experience in embedded systems and Linux driver design.

Course Objective

The Embedded Linux Device Drivers Training Serves the Multiple Objectives:

- Understand the fundamentals of Linux Kernel and Device Drivers
- Learn the complete development workflow for character drivers
- Apply kernel synchronization, process, and interrupt management in real projects
- Build strong debugging, integration, and system-level design skills

Target Group

- Embedded Engineers looking to transition into Linux Device Driver Development
- Developers working on Embedded Linux Products
- Professionals aiming to strengthen their kernel-level programming expertise

Prerequisites

- Knowledge of C Programming
- Basic understanding of Linux OS and command-line environment

Methodology

- Concept lectures blended with live demonstrations
- Structured, progressive hands-on exercises
- Weekly project-oriented assignments
- Mentor feedback and peer review sessions
- Capstone project applying end-to-end driver development

Learning Outcomes

- Navigate and understand Linux kernel source organization
- Write, build, and integrate Linux kernel modules and character drivers
- Implement synchronization mechanisms and wait queues
- Develop drivers handling timers, interrupts, and deferred work
- Apply kernel concepts to real embedded hardware scenarios
- Execute and present an industry-grade driver project

Duration

Typically, it takes 2.5 months to complete the course with Assignments, Challenges and Project. But, since concepts are already available in recorded self-paced form, it can completed is less time as well

Course Flow — Learning Journey Structure

Phase 1: Learn the Foundations

Step 1: Go Through the Recorded Sessions

- Watch the self-paced video lessons on kernel internals, modules, character drivers, synchronization, and interrupts.
- Get conceptual clarity through practical demonstrations and guided exercises.

Step 2: Complete the Assignments

- Implement the concepts learned in each module.
- Submit your hands-on assignments for review to strengthen your fundamentals.

Phase 2: Interact & Reinforce

Step 3: Attend Weekly Live Meetups

- Join the instructor-led live sessions every week.
- Discuss common challenges, debug your code live, and gain deeper clarity.

Step 4: Get Assignments Reviewed

- Receive personalized feedback on your implementations.
- Understand best practices, kernel coding standards, and optimization techniques.

Phase 3: Apply & Challenge

Step 5: Take Up the Challenge Assignments (Advanced Level)

- Solve **industry-inspired problem statements** like interrupt-driven GPIO handling, kernel synchronization problems, or driver–user space communication.
- These challenges push participants beyond tutorials into real-world development scenarios.

Phase 4: Integrate & Showcase

Step 6: Participate in the Industry-Grade Project

- Develop a complete Linux driver project integrating multiple kernel subsystems.
- Apply all the learned concepts — from initialization and synchronization to interrupt handling.
- Present your project for review and earn your **Certificate of Proficiency**.

Detailed Course Curriculum

Module 1: BBB Setup & Introduction to Linux Driver

- Setting up BeagleBone Black for Kernel Internals
- Understanding the Linux Driver Ecosystem
- Linux Kernel Source Organization

Hands-On:

- Configure & build the Linux kernel
- Write and build a simple Linux kernel module
- Statically integrate the driver into the kernel

Module 2: Linux Kernel Module

- Kernel module structure and key commands
- Building and inserting kernel modules dynamically

Hands-On:

- Develop your first kernel module
- Explore module parameters and dependency handling

Module 3: Character Driver – Part 1

- Understanding Character Drivers
- Major & Minor Numbers
- Registering and Unregistering a Driver
- Building the First Character Driver

Hands-On:

- Write and test a basic character driver
- Enhance it to implement file operations

Module 4: Character Driver – Part 2

- Exchanging data between kernel and user space
- Introduction to `udev` and automatic device file creation
- Controlling GPIOs and implementing `ioctl` operations

Hands-On:

- Data transfer between kernel and user-space applications
- Auto-loading the driver in an Embedded Linux system
- Implement GPIO-based LED control
- Extend driver functionality using `ioctl`

Module 5: Kernel Process Management

- Process synchronization mechanisms: Mutex, Semaphore, and Spinlock
- Sleeping and waking up processes
- Wait Queues and synchronization between processes

Hands-On:

- Producer–Consumer driver using synchronization primitives
- Implement wait queues for event-based synchronization

Module 6: Kernel Timing Management

- Kernel timing architecture and “Jiffies”
- Implementing kernel timers and understanding timing mechanisms

Hands-On:

- Write a driver to demonstrate periodic actions using kernel timers

---

Module 7: Interrupt Management & Deferred Work

- Understanding interrupts and their necessity
- Interrupt registration and handling in Linux
- SoftIRQs and Bottom Halves
- Tasklets and Work Queues

Hands-On:

- Develop a driver to handle hardware interrupts
- Implement tasklets and work queues as bottom halves

Module 8: Wrap-Up & Next Steps

- Review of key concepts
- Capstone project overview

- Guidance on industry applications and career progression

Capstone Project (Industry-Grade)

A comprehensive end-to-end driver development project that consolidates all topics:

- Driver design, implementation, and integration
- User-space testing and validation