ORIGIN ACADEMY · COURSE SYLLABUS

Embedded Linux & Kernel Developer

Build embedded Linux systems from the bootloader up, then write kernel drivers: toolchains, U-Boot, the kernel and Device Tree, root filesystems, Buildroot and Yocto, drivers, debugging and real-time.

16 weeks · about 12 hours a week · live cohort of up to 10
Curriculum last updated 19 November 2026

Availability

This course opens soon. Join the waitlist on the course page to hear first about dates and prices.

Before you start

- C: pointers, structs, function pointers and the preprocessor (our C Programming course)
- The Linux command line (our Linux Administration course, or equivalent)
- Embedded basics help: our Embedded Systems Engineer course is ideal
- English good enough to follow technical documentation and live sessions: about a credit (C6 or better) in WASSCE or NECO English Language, or an equivalent such as KCSE or IGCSE English (roughly CEFR level B1). You don't need the certificate: Phase 0 is free, so you can check for yourself
- A laptop running Linux, macOS, or Windows with WSL

How the course works

- A live cohort of up to 10 students, with a mentor.
- At least half your time is hands-on: every module ends with a lab.
- Live sessions are recorded, so you can rewatch them or catch up.
- Questions between sessions are answered within 24 hours on weekdays.
- A mentor reviews every phase project, like a pull request at work.

Phase 0 is free for everyone, before you enroll.

Curriculum

PHASE 0 · WEEKS 1–2 · FREE

Embedded Linux Architecture & Toolchains

How embedded Linux systems fit together, cross-compiling toolchains and C libraries, and a lab you can run on any laptop with QEMU.

- How an embedded Linux system fits together
- Hardware platforms and the development environment
- Cross-compiling toolchains and C libraries: glibc, musl and uClibc-ng
- Building a toolchain with Crosstool-NG
- Your lab: QEMU, and real boards if you have one

You build: Cross-compiling toolchain built with Crosstool-NG

PHASE 1 · WEEK 3

Bootloaders & the Boot Flow

How ARM and x86 systems boot, Trusted Firmware-A, U-Boot, and booting over the network.

- Boot flows on ARM and x86
- U-Boot: build, install, commands and environment
- Trusted Firmware-A
- Network boot with TFTP

You build: Board that boots from U-Boot over TFTP

PHASE 2 · WEEKS 4–5

The Kernel & Device Tree

Kernel architecture, choosing a kernel, configuring and cross-compiling it, the Device Tree, and booting with a root filesystem over NFS.

- Kernel architecture and user-space interfaces
- Mainline, vendor and LTS kernels
- Configuring and cross-compiling the kernel
- The Device Tree
- Booting with an NFS root filesystem

You build: Kernel built and booted over the network

PHASE 3 · WEEKS 6–7

Root Filesystems & Storage

Building a root filesystem from scratch with BusyBox, block and flash filesystems, and cross-compiling real libraries and applications.

- Root filesystem layout and device files
- BusyBox and init
- Block filesystems: ext4, SquashFS and tmpfs
- Flash storage: MTD and UBIFS
- Cross-compiling libraries and applications

You build: Root filesystem built from scratch

PHASE 4 · WEEKS 8–9

Build Systems: Buildroot & Yocto

Why build systems, Buildroot, the Yocto Project, system stacks like systemd and udev, and open-source licence compliance.

- Build systems vs binary distributions
- Buildroot: configuration, packages and images
- The Yocto Project: layers, recipes and BitBake
- systemd, udev and D-Bus
- Open-source licences and compliance

You build: Reproducible images with Buildroot and Yocto

PHASE 5 · WEEKS 10–11

Kernel Modules & the Device Model

Writing kernel modules, Device Tree bindings and pin muxing, the device model and the I2C subsystem, starting a driver you'll grow for the rest of the course.

- Writing kernel modules
- Device Tree bindings and pin muxing
- The Linux device model
- The I2C subsystem
- Running project: a driver for an I2C sensor

You build: I2C sensor driver, part 1

PHASE 6 · WEEKS 12–13

Kernel Frameworks & Drivers

Kernel frameworks such as input and IIO, memory-mapped I/O and platform drivers, interrupts, locking and DMA.

- Kernel frameworks: input and IIO
- Memory-mapped I/O and platform drivers
- Interrupts, sleeping and wait queues
- Locking and concurrency in the kernel
- DMA

You build: I2C sensor driver, part 2: interrupt-driven and exposed through a kernel framework

PHASE 7 · WEEK 14

Debugging, Real-Time & Performance

Debugging the kernel, tracing and profiling, real-time Linux with PREEMPT_RT, boot time and power management.

- Kernel debugging: printk, dynamic debug and oops analysis
- Tracing and profiling with ftrace and perf
- Real-time Linux with PREEMPT_RT
- Boot time and power management

You build: Profiling report and boot-time reduction

PHASE 8 · WEEKS 15–16

Capstone

Bring up Linux on a board to an enterprise brief: board support package, custom driver and an upstream-quality patch.

- BSP design
- Board bring-up
- Preparing an upstream-quality patch
- Demo and review

You build: Board support package with a custom driver

Your certificate

You earn it by doing two things:

- Attend every live session. If you miss one, make it up by watching the recording and submitting that week's lab within 7 days. You can do this up to 2 times per course.
- Score more than 50% on the final evaluation at the end of the course.

Your Enterprise Readiness Rating is separate: it scores your projects, capstone and assessments so employers can see how job-ready you are.

Questions

Email academy@originclouds.io, or see the course online: https://academy.originclouds.io/courses/embedded-linux-kernel-developer