BÀI 6: QUY TRÌNH KHỞI ĐỘNG LINUX

Linux Boot Sequence & Bootloader Overview

🎯 Mục tiêu bài học

1. **Big Picture:** Hiểu rõ 4 giai đoạn khởi động (ROM → SPL → U-Boot → Kernel).
2. **Tại sao:** Tại sao cần chia Bootloader làm 2 phần (SPL và U-Boot)?
3. **Phân biệt:** Boot process trên PC (BIOS/UEFI) khác gì với Embedded Board.
4. **Thực hành:** Đọc hiểu log khởi động (Boot logs) để debug.

1. So sánh: PC vs Embedded

Giai đoạn	Máy tính (PC/Laptop)	Embedded (RPi, BeagleBone)
Phần cứng	Có sẵn BIOS/UEFI trên Flash riêng.	Không có BIOS. Chỉ có ROM Code trong chip.
Khởi động	BIOS tìm ổ cứng → Grub → Linux.	ROM Code tìm thẻ nhớ/eMMC → U-Boot → Linux.
Cấu hình	Giao diện đồ họa (F2/Del).	Dòng lệnh (Command Line) qua UART.
Độ khó	Dễ (Cài Win/Linux là xong).	Khó (Phải tự build Bootloader cho từng loại RAM/Chip).

2. Quy trình khởi động 4 bước

1. **ROM Code:** Cứng, không sửa được.
2. **SPL (Secondary Program Loader):** Khởi tạo RAM.
3. **TPL (Third Program Loader) / U-Boot:** Bootloader chính.
4. **Linux Kernel:** Hệ điều hành.

Giai đoạn 1: ROM Code (Boot ROM)

- **Nơi chứa:** Được nhà sản xuất chip (TI, STM, Broadcom) nạp cứng vào Silicon. Không thể thay đổi.
- **Nhiệm vụ:**
  1. Chạy ngay khi có điện (Power On).
  2. Thiết lập Clock cơ bản.
  3. **Quan trọng:** Tìm kiếm file Bootloader (SPL) từ các thiết bị ngoại vi (SD Card, eMMC, USB, UART) theo thứ tự ưu tiên (Boot Order).
  4. Copy SPL vào bộ nhớ đệm nội (SRAM).

> *Ví dụ: Trên BeagleBone, nhấn giữ nút S2 để ép ROM Code tìm boot từ thẻ nhớ trước eMMC.*

Giai đoạn 2: SPL (Secondary Program Loader)

- **Vấn đề:** ROM Code rất "ngu", nó không biết thanh RAM DDR3 bên ngoài hoạt động thế nào. SRAM nội bộ thì quá nhỏ (< 128KB), không đủ chứa U-Boot khổng lồ.
- **Giải pháp:** Cần một chương trình nhỏ gọn (SPL) chạy trước.
- **Nhiệm vụ:**
  1. Khởi tạo bộ điều khiển RAM (DDR Controller).
  2. Thiết lập Clock hệ thống lên tốc độ cao.
  3. Tải **U-Boot** chính từ thẻ nhớ vào RAM (DRAM).
- **Tên file:** Thường là `MLO` (trên TI) hoặc `u-boot-spl.bin`.

Giai đoạn 3: U-Boot (Das U-Boot)

Đây là "Ông trùm" của Bootloader trong thế giới Embedded.

- **Nhiệm vụ:**
  1. Cung cấp giao diện dòng lệnh (CLI) để người dùng cấu hình.
  2. Hỗ trợ đọc file từ FAT32, EXT4, TFTP, NFS.
  3. Load **Linux Kernel** (`zImage`) và **Device Tree** (`.dtb`) vào RAM.
  4. Truyền tham số boot (`bootargs`) cho Kernel.
  5. Nhảy tới Kernel (`bootz`).

Giai đoạn 4: Linux Kernel & RootFS

Khi U-Boot trao quyền (`Starting kernel...`), Kernel làm gì?

1. **Kernel Init:** Nhận diện phần cứng (CPU, Memory).
2. **Driver Init:** Load driver tích hợp sẵn.
3. **Mount RootFS:** Tìm và gắn phân vùng chứa file hệ thống (`/`).
4. **Run Init:** Chạy chương trình đầu tiên của User Space (`/sbin/init` hoặc `/bin/sh`).

- Nếu Init chết $\rightarrow$ **Kernel Panic** (Màn hình xanh chết chóc của Linux).

🛠️ PHẦN THỰC HÀNH (LAB 06)

Phân tích Log khởi động (Boot Logs Anatomy)

Phân tích Log (BeagleBone Example)

```
U-Boot SPL 2019.04 (Stage 2) <-- SPL chạy
Trying to boot from MMC1
U-Boot 2019.04 (Stage 3) <-- U-Boot chính chạy
CPU : AM335X-GP rev 2.1, 1000 MHz
DRAM: 512 MiB <-- RAM đã nhận
MMC: OMAP SD/MMC: 0, OMAP SD/MMC: 1
Net: cpsw, usb_ether
Hit any key to stop autoboot: 0 <-- Đếm ngược
```

> **Yêu cầu:** Kết nối board thật qua UART, Reset board và quan sát log. Xác định đâu là đoạn của SPL, đâu là U-Boot, đâu là Kernel.

Bài tập tình huống (Case Study)

Tình huống: Board khởi động lên, in ra dòng sau rồi đứng im:

```
CCC... *(Trên dòng chip TI OMAP/Sitara)*
```

Nguyên nhân là gì?

- A. Hỏng RAM.
- B. ROM Code không tìm thấy thẻ nhớ/eMMC (Boot failed).
- C. Kernel bị lỗi.

> **Đáp án:** B. Chữ 'C' là tín hiệu ROM Code gửi ra UART để chờ nạp qua cổng Serial (Xmodem) khi các nguồn boot khác thất bại.

📝 Bài tập về nhà

1. Tìm hiểu về biến môi trường `bootargs` trong U-Boot. Nó dùng để làm gì?
2. Tải source code U-Boot mới nhất từ trang chủ `denx.de` hoặc GitHub.
3. Cấu hình Toolchain để chuẩn bị cho **Bài 7: Build U-Boot**.

Q & A

Hẹn gặp lại ở Bài 7: Thực hành Build U-Boot!