BÀI 4: TOOLCHAIN & CROSS-COMPILATION

Bộ công cụ biên dịch chéo

🎯 Mục tiêu bài học

1. **Hiểu bản chất:** Tại sao cần Cross-Compilation? (x86 vs ARM).
2. **Giải phẫu Toolchain:** Toolchain không chỉ là Compiler. Nó gồm những gì?
3. **Quy ước đặt tên:** Đọc hiểu ý nghĩa của `arm-linux-gnueabihf`.
4. **Thực hành:** Cài đặt Toolchain, cấu hình biến môi trường và "mổ xẻ" file thực thi.

1. Native vs Cross Compilation

Đặc điểm	Native Compilation	Cross Compilation
Nơi Build	Máy tính (PC)	Máy tính (Host - x86)
Nơi Chạy	Máy tính (PC)	Board nhúng (Target - ARM)
Tại sao dùng?	Phát triển Desktop App	Board nhúng quá yếu để tự build code cho chính nó.
Compiler	`gcc`	`arm-linux-gnueabihf-gcc`

> **Ví dụ:** Bạn không thể dùng lò vi sóng (Board) để sản xuất ra cái lò vi sóng. Bạn cần nhà máy (PC) để làm việc đó.

2. Thành phần của Toolchain

Một Toolchain đầy đủ gồm 4 thành phần chính:

1. **Binutils:** Bộ công cụ thao tác file nhị phân (`ld` , `as` , `objcopy` , `strip` , `readelf`).
2. **C Library (C Runtime):** Thư viện chuẩn kết nối code C với Kernel.
   - *glibc:* Chuẩn, đầy đủ nhưng nặng (Dùng cho PC/Server).
   - *uClibc / musl:* Nhẹ, tối ưu cho Embedded.
3. **Kernel Headers:** Các file `.h` định nghĩa API của Linux Kernel.
4. **Compiler (GCC/Clang):** Dịch code C sang Assembly/Machine Code.

3. Quy ước đặt tên (Naming Convention)

Tên Toolchain thường tuân theo quy tắc: `arch-vendor-os-abi`

Ví dụ: `arm-linux-gnueabihf-gcc`

- `arm` : Kiến trúc CPU (Architecture).
- `linux` : Hệ điều hành chạy trên Target (Kernel).
- `gnu` : Nhà cung cấp (Vendor - thường là GNU hoặc rỗng).
- `eabihf` : ABI (Application Binary Interface).
  - `eabi` : Embedded ABI.
  - `hf` (Hard Float): Dùng bộ xử lý toán học phần cứng (FPU) -> Nhanh.
  - Không có `hf` (Soft Float): Giả lập toán học bằng phần mềm -> Chậm.

4. Các công cụ "ngầm" trong Binutils

Ngoài `gcc`, bạn cần biết các tool sau để debug:

- `ld` **(Linker):** Liên kết các file `.o` thành file chạy.
- `objdump`: Disassemble file chạy ra mã Assembly (để debug).
  - `arm-linux-objdump -d my_app`
- `readelf`: Xem thông tin Header của file ELF.
- `strip`: Xóa bỏ thông tin debug để giảm dung lượng file (Quan trọng khi làm sản phẩm thương mại).
- `nm`: Liệt kê các Symbol (tên hàm, biến) trong file.

5. C Library: Glibc vs Musl/uClibc

Thư viện	Đặc điểm	Ứng dụng
glibc	Chuẩn GNU, tương thích cao, rất nặng (vài MB).	Desktop, RPi, Server.
uClibc-ng	Cực nhẹ, hỗ trợ kiến trúc lạ tốt.	Router, Camera cũ.
musl	Hiện đại, tuân thủ chuẩn POSIX nghiêm ngặt, nhẹ.	Alpine Linux, IoT hiện đại.

> **Lưu ý:** Code biên dịch bằng `glibc` sẽ KHÔNG chạy được trên hệ thống dùng `musl` và ngược lại.

🛠️ PHẦN THỰC HÀNH (LAB 04)

Cài đặt & Sử dụng Toolchain Linaro

Bước 1: Cài đặt Toolchain

Trên Ubuntu, chúng ta cài bản Linaro GCC tiêu chuẩn:

```
# Cài đặt GCC cho ARM 32-bit (RPi 2/3, BeagleBone)
sudo apt update
sudo apt install gcc-arm-linux-gnueabihf

# (Optional) Cài đặt cho ARM 64-bit (RPi 4/5)
sudo apt install gcc-aarch64-linux-gnu
```

Kiểm tra phiên bản:

```
arm-linux-gnueabihf-gcc --version
```

Bước 2: Thiết lập biến môi trường

Thay vì gõ dài dòng, ta gán vào biến `CROSS_COMPILE`.

```
# Gõ lệnh này trong Terminal (hoặc thêm vào ~/.bashrc)
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-

# Kiểm tra lại
echo $CROSS_COMPILE
```

> **Tại sao có dấu gạch ngang `-` ở cuối?**
> Để khi ghép với `gcc`, nó thành `arm-linux-gnueabihf-gcc`.

Bước 3: Biên dịch & Mổ xẻ (Dissecting)

1. **Biên dịch:**

```
$CROSS_COMPILEgcc main.c -o main_arm
gcc main.c -o main_pc
```

2. **So sánh file:** Dùng lệnh `file`

```
file main_pc  # Output: ELF 64-bit LSB executable, x86-64...
file main_arm # Output: ELF 32-bit LSB executable, ARM, EABI5...
```

3. **Xem thư viện phụ thuộc:**

```
$CROSS_COMPILEreadelf -d main_arm  # Tìm dòng "Shared library: [libc.so.6]" -> Nó cần glibc.
```

Bước 4: Tối ưu dung lượng (Strip)

Thử so sánh dung lượng trước và sau khi `strip`.

```
# Xem dung lượng gốc
ls -lh main_arm

# Loại bỏ thông tin debug (Symbol table)
$CROSS_COMPILEstrip main_arm

# Xem lại dung lượng
ls -lh main_arm
```

> **Kết quả:** Dung lượng giảm đáng kể (thường 30-50% với file nhỏ).

📝 Bài tập về nhà

1. Tải một Toolchain khác (ví dụ từ trang chủ ARM Developer hoặc Bootlin) thay vì dùng `apt`.
2. Viết Makefile nâng cao: Tự động phát hiện xem biến `CROSS_COMPILE` có được set hay chưa. Nếu chưa -> Báo lỗi.
3. Tìm hiểu sự khác biệt giữa **Static Linking** (`-static`) và **Dynamic Linking**.
    - *Gợi ý:* Build thử với cờ `-static` và so sánh dung lượng.

Q & A

Hẹn gặp lại ở Bài 5: Thư viện tĩnh & Động!