Name: Đào Đình Tú

ID: EEACIU17053

**Report Embedded Linux**

1. **Choosing Linux**

• Microprocessors have a memory management unit (MMU), and microcontrollers don’t. Yes, [you can run Linux without an MMU](https://en.wikipedia.org/wiki/%CE%9CClinux).

• But you usually shouldn’t. They’re power-hungry: microcontrollers are built on larger processes than application processors to reduce their leakage current. And without an MMU and generally-low clock speeds, they’re downright slow.

• Other than the MMU, the lines between MCUs and MPUs are getting blurred.

1. **When not to choose Linux**

• Phần cứng của bạn có hoạt động tốt không? So với hệ điều hành thời gian thực truyền thống (RTOS) như VxWorks hay QNX, Linux đòi hỏi nhiều tài nguyên hơn. Nó cần ít nhất một bộ xử lý 32-bit và nhiều bộ nhớ hơn.

• Hệ thống của bạn có là system real-time? Linux có thể xử lý nhiều hoạt động thời gian thực miễn là bạn chú ý đến một số chi tiết nhất định.

• Mã của bạn có yêu cầu phê duyệt theo quy định (y tế, ô tô, hàng không vũ trụ, v.v.) không? Gánh nặng của việc xác minh và xác thực theo quy định có thể khiến một hệ điều hành khác trở thành lựa chọn tốt hơn.

1. **Meeting the players**

Những main players như sau:

• **The open source community**: Cộng đồng mã nguồn mở: Rốt cuộc, đây là động cơ tạo ra phần mềm mà bạn sẽ sử dụng. Cộng đồng là một liên minh lỏng lẻo của các nhà phát triển, nhiều người trong số họ được tài trợ theo một cách nào đó, có lẽ bởi một tổ chức phi lợi nhuận, một tổ chức học thuật, hoặc một công ty thương mại. Họ làm việc cùng nhau để đạt được mục tiêu của các dự án khác nhau. Có rất nhiều trong số chúng — một số nhỏ, một số lớn. Một số mà chúng tôi sẽ sử dụng trong phần còn lại của cuốn sách này là bản thân Linux, U-Boot, BusyBox, Buildroot, Dự án Yocto và nhiều dự án dưới sự bảo trợ của GNU.

• **CPU architects**: Đây là những tổ chức thiết kế CPU mà chúng tôi sử dụng. Các những cái quan trọng ở đây là Arm / Linaro (Arm Cortex-A), Intel (x86 và x86\_64), SiFive (RISC-V) và IBM (PowerPC). Họ thực hiện hoặc ít nhất, ảnh hưởng đến hỗ trợ cho kiến trúc CPU cơ bản.

• **SoC (System on a chip) vendors** (Broadcom, Intel, Microchip, NXP, Qualcomm, TI và nhiều nhà cung cấp khác): Họ lấy hạt nhân và chuỗi công cụ từ các kiến trúc sư CPU và sửa đổi chúng để hỗ trợ chip của họ. Họ cũng tạo ra các bảng tham chiếu: các thiết kế được cấp sau sử dụng để tạo bảng phát triển và các sản phẩm đang hoạt động.

• **Board vendors and OEMs**: Những người này lấy các thiết kế tham khảo từ các SoC vendors và xây dựng chúng vào các sản phẩm cụ thể. Vd: Các bo mạch phát triển giá rẻ như BeagleBoard / BeagleBone và Raspberry Pi đã tạo ra hệ sinh thái phần mềm và phần cứng bổ trợ của riêng họ.

• **Commercial Linux vendors**: Các công ty như Siemens (Mentor), Timesys và Wind River cung cấp các bản phân phối Linux thương mại đã trải qua quá trình xác minh và xác nhận theo quy định nghiêm ngặt trên nhiều ngành (y tế, ô tô, hàng không vũ trụ, v.v.).

\*\* Bạn không thể chỉ lấy hạt nhân mới nhất từ ​​https://www.kernel.org/, trừ một số trường hợp hiếm hoi, vì nó không hỗ trợ chip hoặc bo mạch mà bạn đang sử dụng.

1. **The four elements of embedded Linux**

Bốn yếu tố của Embedded Linux:

• Toolchain: The compiler and other tools needed to create code for your target device.  
• Bootloader: Chương trình khởi tạo bo mạch và tải nhân Linux.  
• Kernel: Đây là trái tim của hệ thống, quản lý tài nguyên hệ thống và giao tiếp với phần cứng.  
• Root filesystem: Chứa các thư viện và chương trình được chạy khi nhân đã hoàn thành khởi tạo.

1. **Selecting hardware for embedded Linux**

• Đầu tiên, một CPU architecuture được hỗ trợ bởi kernel. Những thứ thường được tìm thấy nhất trong các thiết bị nhúng là Arm, MIPS, PowerPC và x86, mỗi biến thể trong 32 và 64-bit, tất cả đều có đơn vị quản lý bộ nhớ (MMU).

• Thứ hai, bạn sẽ cần một dung lượng RAM hợp lý.

• Third, there is non-volatile storage, usually flash memory. Linux có hỗ trợ rộng rãi cho các thiết bị lưu trữ flash, bao gồm chip flash thô NOR và NAND, và managed flash dưới dạng thẻ SD, chip eMMC, USB flash memory, v.v.

• Thứ tư, một cổng nối tiếp rất hữu ích, tốt nhất là một cổng nối tiếp dựa trên UART.

• Thứ năm, bạn cần một số phương tiện tải phần mềm khi bắt đầu từ đầu. Các SoC hiện đại cũng có khả năng tải mã khởi động trực tiếp từ phương tiện di động, đặc biệt là thẻ SD và micro SD, hoặc các giao diện nối tiếp như UART hoặc USB.

1. **Obtaining the hardware**

Ba thiết bị điển hình: Raspberry Pi 4, BeagleBone Black và QEMU.

Sử dụng Raspberry Pi 4 và BeagleBone Black, khi cần tương tác với phần cứng thực và nhìn thấy đèn LED thực sự nhấp nháy.

QEMU là 1 trình giả lập có thể được sử dụng để tạo ra một loạt các hệ thống đặc trưng cho phần cứng nhúng.

+ QEMU

QEMU là một trình giả lập máy. Có thể mô phỏng một kiến trúc bộ xử lý và một số bo mạch được xây dựng bằng kiến trúc đó. Ví dụ, chúng tôi có những điều sau:

• qemu-system-arm: 32-bit Arm

• qemu-system-mips: MIPS

• qemu-system-ppc: PowerPC

• qemu-system-x86: x86 và x86\_64

Đối với mỗi kiến trúc, QEMU mô phỏng một loạt phần cứng mà bạn có thể thấy bằng cách sử dụng tùy chọn trợ giúp -machine. Mỗi máy mô phỏng hầu hết phần cứng thường được tìm thấy trên bo mạch đó.

Đây là một ví dụ cụ thể:

**$ qemu-system-arm -machine vexpress-a9 -m 256M -drive   
file=rootfs.ext4,sd -net nic -net use -kernel zImage -dtb   
vexpress- v2p-ca9.dtb -append "console=ttyAMA0,115200 root=/  
dev/mmcblk0" -serial stdio -net nic,model=lan9118 -net   
tap,ifname=tap0**

Các tùy chọn được sử dụng trong dòng lệnh trước như sau:

• -machine vexpress-a9: Tạo mô phỏng của bảng phát triển Arm Verspose Express với bộ xử lý Cortex A-9

• -m 256M: Tích hợp nó với 256 MiB RAM

• -drive file = rootfs.ext4, sd: Kết nối giao diện SD với tệp cục bộ rootfs.ext4 (chứa hình ảnh hệ thống tệp)

• -kernel zImage: Tải nhân Linux từ tệp cục bộ có tên zImage

• -dtb vexpress-v2p- ca9.dtb: Tải device tree từ tệp cục bộ vexpress-v2p-ca9.dtb

• -append "...": Thêm chuỗi này làm dòng lệnh hạt nhân.

• -serial stdio: Kết nối cổng nối tiếp với thiết bị đầu cuối khởi chạy QEMU, thông thường để bạn có thể đăng nhập vào máy mô phỏng thông qua bảng điều khiển nối tiếp.

• -net nic, model = lan9118: Tạo giao diện mạng.

• -net tap, ifname = tap0: Kết nối giao diện mạng với giao diện mạng ảo, tap0.

**Learning about Toolchains**

Chuỗi công cụ là yếu tố đầu tiên của Linux nhúng và là điểm khởi đầu của dự án. Sẽ sử dụng nó để biên dịch tất cả mã sẽ chạy trên thiết bị.

1. **Technical requirements**

• A Linux-based host system with *autoconf, automake, bison, bzip2, cmake, flex, g++, gawk, gcc, gettext, git, gperf, help2man, libncurses5-dev, libstdc++6, libtool, libtool-bin, make, patch, python3-dev, rsync, texinfo, unzip, wget, and xz-utils* or their equivalents installed.

I recommend using Ubuntu 20.04 LTS or later since the exercises in this chapter were  
all tested against that Linux distribution at the time of writing. Here is the command to  
install all the required packages on Ubuntu 20.04 LTS:  
**$ sudo apt-get install autoconf automake bison bzip2 cmake flex g++ gawk gcc  
gettext git gperf help2man libncurses5-dev libstdc++6 libtool libtool-bin make  
patch python3-dev rsync texinfo unzip wget xz-utils**

1. **Introducing toolchains**

Chuỗi công cụ GNU tiêu chuẩn bao gồm ba thành phần chính:

• **Binutils**: Một tập hợp các tiện ích nhị phân bao gồm the assembler và the linker .

• **GNU Compiler Collection (GCC)**: Đây là các trình biên dịch cho C và các ngôn ngữ khác, tùy thuộc vào phiên bản của GCC, bao gồm C ++, Objective-C, Objective-C ++, Java, Fortran, Ada và Go.

• **C library**: là một bộ các API chuẩn dựa trên POSIX, các ứng dụng sử dụng C library để giao tiếp với kernel.

+ Types of toolchains

Có hai loại chuỗi công cụ:

• Native toolchain: Loại toolchain chạy trên hệ thống giống với target device (device dùng để run chương trình build ra bởi toolchain).

• Cross toolchain: Loại toolchain này chạy trên hệ thống khác với target device, ví dụ dùng desktop để build ra chương trình chạy trên một thiết bị nhúng khác.

Hầu hết tất cả quá trình phát triển Linux nhúng đều được thực hiện bằng cách sử dụng chuỗi công cụ phát triển chéo, một phần vì hầu hết các thiết bị nhúng không phù hợp lắm với việc phát triển chương trình vì chúng thiếu sức mạnh tính toán, bộ nhớ và lưu trữ, nhưng cũng vì nó giữ môi trường máy chủ và môi trường đích riêng biệt. Điểm thứ hai đặc biệt quan trọng khi máy chủ và máy chủ đích đang sử dụng cùng một kiến trúc, ví dụ: x86\_64. Trong trường hợp này, thật hấp dẫn để biên dịch nguyên bản trên máy chủ và chỉ cần sao chép các tệp nhị phân vào đích.

+ CPU architectures

Các CPU khác nhau sẽ cần các toolchain khác nhau để build ra chương trình có thể run được trên CPU đó. Các yếu tố sau của CPU sẽ ảnh hưởng lên toolchain tương ứng:

• **CPU architecture**: ARM, (MIPS), x86\_64, v.v.

• **Big- or little-endian operation**.

• **Floating point support**: Không phải processor nào cũng khối hỗ trợ tính toán floating point bằng hardware. Đối với CPU mà processor của nó không có hardware floating point thì toolchain cần cấu hình để dùng thư viện software floating point.

• **Application Binary Interface** (**ABI**): Quy ước gọi được sử dụng để chuyển các tham số giữa các lệnh gọi hàm. Có thể là abi, eabi, eabihf

Tên của mỗi công cụ trong chuỗi công cụ, xác định các kết hợp khác nhau có thể được tạo ra. Nó bao gồm một bộ ba hoặc bốn thành phần được phân tách bằng dấu gạch ngang, như được mô tả ở đây:

• **CPU**: Đây là kiến trúc CPU, chẳng hạn như ARM, MIPS hoặc x86\_64.

• **Vendor**: Nhà cung cấp/ nhà phát triển ra toolchain. Ví dụ bao gồm buildroot, poky hoặc không xác định.

• **Kernel**: Luôn luôn là “linux” với toolchain cho Linux.

• **Operating system**: *gnueabi, gnueabihf, musleabi* hoặc *musleabihf, uclibcgnueabi, hoặc uclibcgnueabihf*

You can find the tuple used when building the toolchain by using the -dumpmachine  
option of gcc. For example, you may see the following on the host computer:  
**$ gcc -dumpmachine  
x86\_64-linux-gnu**

This tuple indicates a CPU of x86\_64, a kernel of linux, and a user space of gnu.

1. **Choosing the C library**

There are several C libraries to choose from. The main options are as follows:

• glibc: Đây là thư viện GNU C chuẩn, có tại https://gnu.org/software/libc. Nó lớn và cho đến gần đây, không thể cấu hình được, nhưng nó là bản triển khai hoàn chỉnh nhất của API POSIX. Giấy phép là LGPL 2.1.

• musl libc: Có sẵn tại https://musl.libc.org. Thư viện musl libc tương đối mới nhưng đã được nhiều người chú ý như một giải pháp thay thế nhỏ và tuân thủ các tiêu chuẩn cho GNU libc. Đây là một lựa chọn tốt cho các hệ thống có dung lượng RAM và dung lượng lưu trữ hạn chế. Nó có giấy phép MIT.

• uClibc-ng: Tính năng này có tại https://uclibc-ng.org. u thực sự là một ký tự mu Hy Lạp, chỉ ra rằng đây là thư viện C của vi điều khiển. Nó lần đầu tiên được phát triển để làm việc với uClinux (Linux cho CPU không có bộ quản lý bộ nhớ) nhưng sau đó đã được điều chỉnh để sử dụng với Linux đầy đủ. Thư viện uClibc-ng là một nhánh của dự án uClibc ban đầu (https://uclibc.org), không may đã bị hỏng. Cả hai đều được cấp phép với LGPL 2.1.

• eglibc: Cái này có tại http://www.eglibc.org/home. Bây giờ đã lỗi thời, eglibc là một nhánh của glibc với những thay đổi để làm cho nó phù hợp hơn với việc sử dụng nhúng. Trong số những thứ khác, eglibc đã bổ sung các tùy chọn cấu hình và hỗ trợ cho các kiến ​​trúc không được glibc đề cập, đặc biệt là lõi CPU PowerPC e500. Cơ sở mã từ eglibc đã được hợp nhất trở lại thành glibc trong phiên bản 2.20. Thư viện eglibc không còn được duy trì.

Vì vậy, mà để chọn? Lời khuyên của tôi là chỉ sử dụng uClibc-ng nếu bạn đang sử dụng uClinux. Nếu bạn có số lượng bộ nhớ hoặc RAM rất hạn chế, thì musl libc là một lựa chọn tốt, nếu không, hãy sử dụng glibc, như được hiển thị trong biểu đồ sau:

1. **Finding a toolchain**

• Một SoC hoặc nhà cung cấp bo mạch. Hầu hết các nhà cung cấp đều cung cấp chuỗi công cụ Linux.

• Một tập đoàn chuyên cung cấp hỗ trợ cấp hệ thống cho một kiến ​​trúc nhất định. Ví dụ: Linaro, (https://www.linaro.org) có các công cụ được tạo sẵn cho kiến ​​trúc ARM.

• Nhà cung cấp công cụ Linux bên thứ ba, chẳng hạn như Mentor Graphics, TimeSys hoặc MontaVista.

• Các gói công cụ chéo dành cho bản phân phối Linux dành cho máy tính để bàn của bạn. Ví dụ, các bản phân phối dựa trên Debian có các gói để biên dịch chéo cho các mục tiêu ARM, MIPS và PowerPC.

• SDK nhị phân được tạo ra bởi một trong những công cụ xây dựng được nhúng tích hợp. Dự án Yocto có một số ví dụ tại [http://downloads.yoctoproject.org/releases/yocto/yocto-[version]/toolchain](http://downloads.yoctoproject.org/releases/yocto/yocto-%5bversion%5d/toolchain).

• Một liên kết từ một diễn đàn mà bạn không thể tìm thấy nữa.

Một giải pháp thay thế đơn giản hơn là sử dụng crosstool-NG, gói quy trình này thành một tập hợp các tập lệnh và có giao diện người dùng hướng menu.

Đơn giản hơn vẫn là sử dụng một hệ thống xây dựng như Buildroot hoặc Dự án Yocto, vì chúng tạo chuỗi công cụ như một phần của quá trình xây dựng.

Với sự phát triển vượt bậc của crosstool-NG, việc xây dựng chuỗi công cụ của riêng bạn chắc chắn là một lựa chọn hợp lệ và khả thi. Hãy xem cách thực hiện điều đó tiếp theo.

1. **Building a toolchain using crosstool-NG**

We will use crosstool-NG to build toolchains for the BeagleBone Black and QEMU.

Installing crosstool-NG  
Trước khi có thể tạo crosstool-NG từ nguồn, trước tiên bạn cần cài đặt chuỗi công cụ gốc và một số công cụ xây dựng trên máy chủ của mình.  
Tiếp theo, lấy bản phát hành hiện tại từ kho lưu trữ crosstool-NG Git. Trong các ví dụ của tôi, tôi đã sử dụng phiên bản 1.24.0. Giải nén nó và tạo hệ thống menu giao diện người dùng, ct-ng, như được hiển thị trong các lệnh sau:  
**$ git clone https://github.com/crosstool-ng/crosstool-ng.git  
$ cd crosstool-ng  
$ git checkout crosstool-ng-1.24.0  
$ ./bootstrap  
$ ./configure --prefix=${PWD}  
$ make  
$ make install**  
  
The --prefix=${PWD} option means that the program will be installed into the current  
directory, which avoids the need for root permissions, as would be required if you were to  
install it in the default location /usr/local/share.  
We now have a working installation of crosstool-NG that we can use to build cross  
toolchains with. Type bin/ct-ng to launch the crosstool menu.

+ Building a toolchain for QEMU

On the QEMU target, you will be emulating an ARM-versatile PB evaluation board that has an ARM926EJ-S processor core, which implements the ARMv5TE instruction set. You need to generate a crosstool-NG toolchain that matches the specification.

You begin by running bin/ct-ng list-samples to find a good base configuration to work from. There isn't an exact fit, so use a generic target, arm-unknown-linuxgnueabi. You select it as shown, running distclean first to make sure that there are no artifacts left over from a previous build:  
$ bin/ct-ng distclean  
$ bin/ct-ng arm-unknown-linux-gnueabiAs with the BeagleBone Black, you can review the configuration and make changes using the configuration menu command bin/ct-ng menuconfig. There is only one change necessary:  
 • In **Paths and misc options**, disable **Render the toolchain read-only** (CT\_PREFIX\_DIR\_RO).  
Now, build the toolchain with the command shown here:  
$ bin/ct-ng buildAs before, the build will take about half an hour. The toolchain will be installed in ~/x-tools/arm-unknown-linux-gnueabi.

1. **Anatomy of a toolchain**

Muốn kiểm tra chuỗi công cụ crosstool-NG mà bạn vừa tạo. Các ví dụ sử dụng chuỗi công cụ ARM Cortex A8 được tạo cho BeagleBone Black: *arm-cortex\_a8-linuxgnueabihf-*. Nếu bạn đã xây dựng chuỗi công cụ ARM926EJ-S cho mục tiêu QEMU, thì tiền tố thay thế sẽ là *arm -unknown-linux-gnueabi*.

Chuỗi công cụ ARM Cortex A8 nằm trong thư mục *~ / x-tools / arm-cortex\_a8-linux-gnueabihf / bin*. Trong đó, bạn sẽ tìm thấy trình biên dịch chéo, *arm-cortex\_a8-linux-gnueabihf-gcc*. Để sử dụng nó, bạn cần thêm thư mục vào đường dẫn của mình bằng lệnh sau:

$ PATH=~/x-tools/arm-cortex\_a8-linux-gnueabihf/bin:$PATH

Now you can take a simple helloworld program, which in the C language looks like this:  
#include <stdio.h>  
#include <stdlib.h>  
int main (int argc, char \*argv[]) {  
printf ("Hello, world!\n");  
return 0;  
}

You compile it like this:  
$ arm-cortex\_a8-linux-gnueabihf-gcc helloworld.c -o helloworld

You can confirm that it has been cross-compiled by using the file command to print the  
type of the file:  
**$ file helloworld  
helloworld: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.  
so.3, for GNU/Linux 4.20.8, with debug\_info, not stripped**

1. **The sysroot, library, and header files**

sysroot là một folder của toolchain, nó chứa các file header, thư viện và các file cấu hình cho việc biên dịch chương trình.

Một số folder quan trọng bên trong sysroot:

• lib: Chứa các shared objects cho thư viện C và trình liên kết / tải động, ld-linux

• usr / lib: static library của C library và bất kỳ thư viện nào khác có thể được cài đặt sau đó

• usr / include: header file của tất cả các library

• usr / bin: một số tool, utilitiy của tất cả các library

• usr / share: Được sử dụng để localizarion và internationalization

• sbin: Cung cấp tiện ích ldconfig, được sử dụng để tối ưu hóa đường dẫn tải thư viện

Thông thường, một số trong số này là cần thiết trên máy chủ phát triển để biên dịch chương trình, và một số khác, ví dụ, các thư viện chia sẻ và ld-linux, là cần thiết cho mục tiêu trong thời gian chạy.

+ Other tools in the toolchain

Dưới đây là danh sách các lệnh để gọi các thành phần khác nhau của chuỗi công cụ GNU:

• addr2line: Converts addresses into filenames and line numbers.

• ar: A utility for creating, modifying and extracting from archives.

• as: Đây là trình Assembly GNU.

• c ++ filt: Điều này được sử dụng để phân loại các ký hiệu C ++ và Java.

• cpp: Đây là bộ tiền xử lý C và được sử dụng để mở rộng #define, #include và các chỉ thị tương tự khác. Bạn hiếm khi cần phải sử dụng nó một mình.

• elfedit: Điều này được sử dụng để cập nhật tiêu đề ELF của các tệp ELF.

• g ++: Đây là giao diện người dùng GNU C ++, giả định rằng các tệp nguồn chứa mã C ++.

• gcc: Đây là giao diện người dùng GNU C, giả định rằng các tệp nguồn chứa mã C.

• gcov: Đây là một công cụ bao phủ mã.

• gdb: Đây là trình gỡ lỗi GNU.

• gprof: Đây là một công cụ cấu hình chương trình.

• ld: Đây là trình liên kết GNU.

• nm: Danh sách này liệt kê các ký hiệu từ các tệp đối tượng.

• objcopy: Được sử dụng để sao chép và dịch các tệp đối tượng.

• objdump: Cái này được sử dụng để hiển thị thông tin từ các tệp đối tượng.

• ranlib: Điều này tạo hoặc sửa đổi một chỉ mục trong thư viện tĩnh, làm cho giai đoạn liên kết nhanh hơn.

• readelf: Điều này hiển thị thông tin về các tệp ở định dạng đối tượng ELF.

• size: Phần này liệt kê kích thước phần và tổng kích thước.

• strings: Điều này hiển thị các chuỗi ký tự có thể in được trong tệp.

• strip: Discards symbols.

Bây giờ chúng ta sẽ chuyển bánh răng từ các công cụ dòng lệnh và quay lại chủ đề của thư viện C.

+ Looking at the components of the C library

Thư viện C không phải là một tệp thư viện đơn lẻ. Nó bao gồm bốn phần chính cùng triển khai API POSIX:

• *libc*: Là thành phần chính của C library, chính chứa các hàm POSIX function như *printf, open, close, read, write, v.v.*

• *libm*: Chứa các hàm toán học như cos, exp và log

• *libpthread*: Chứa tất cả các các POSIX function dùng cho thread( có tiền tố là pthread\_)

• *librt*: Chứa các hàm liên quan đến shared memory và asynchronous I/O.

**Learning about Bootloaders**

Khi hệ thống khởi động lần đầu tiên, hoặc reset. Quyền kiểm soát hệ thống sẽ thuộc về reset vector, nó là một đoạn mã assembly được ghi trước bởi nhà sản xuất chip (**Manufaturer** ). Sau đó reset vector sẽ trỏ tới địa chỉ vùng nhớ chứa các đoạn mã khởi động đầu tiên, cụ thể là boot rom. Nếu không có reset vector thì bộ xử lý sẽ không biết nên thực thi bắt đầu từ đâu.