Embedded Book Club

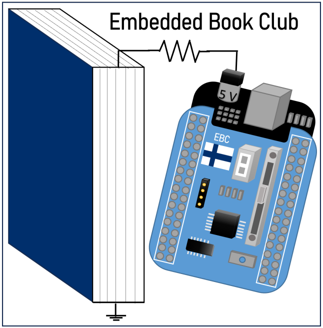
Linux Device Driver Development
Chapter 1: Introduction to Kernel Development

Jonathan Velasco

September 18th, 2023

Embedded Book Club Finland

We're knowledge sharing enthusiasts, focused on hosting in-person events, to bond over technical topics related to embedded systems.

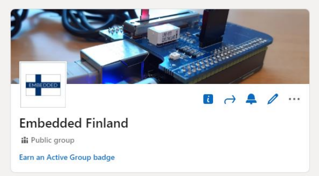


Our goal is to

- Create a community focused on Embedded Systems and related topics
- Share knowledge, and to learn about new topics, trends and practices
- Having fun learning and working on projects together

Introduction

- Who am I? Jonathan Velasco
- Who are you?
- Profession, company?
- Previous experience in Embedded, Linux, etc
- Interests



Chapter 1 - Introduction to Kernel Development

This chapter covers

- Setting up your development environment,
- Configuring the kernel
- Building the kernel.

General information

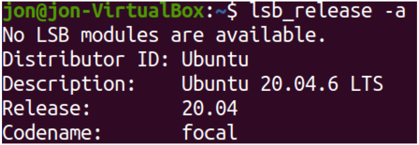
- Linux started as a hobby project in 1991 by Finnish student, Linus Torvalds
- Linux is a must in embedded systems and on servers
- Linux advantages: free of charge, well documented, portable, access to source code, and has a lot of free compatible software

Chapter 1 - Introduction to Kernel Development
Setting up the Development Environment

Embedded Terminology

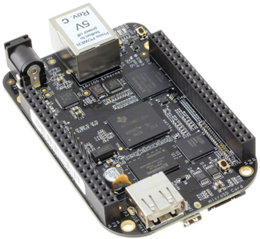
- Target
  - Machine **running** produced binary
- Host
  - Machine **producing** the binary
- Compilation:
  - Native build. Host == Target
- Cross-compilation:
  - Host!= Target

Host Machine - x86



```
jon@jon-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.6 LTS
Release:        20.04
Codename:       focal
```

Target Machine - ARM



Chapter 1 - Introduction to Kernel Development
Setting up the Host Machine

This section applies to Debian systems

Install packages

```
$ sudo apt update
$ sudo apt install gawk wget git diffstat \
    unxip texinfo gcc-multiplib \
    build-essential chrpath socat \
    libsdl1.2-dev xterm ncurses-dev \
    lzop libelf-dev make
```

Chapter 1 - Introduction to Kernel Development
Installing Toolchains

This section covers the installation of tools used for the build process. Set of tools is called Binutils. The term toolchain typically refers to compiler + Binutils + other build-time dependecy libraries

Toolchain naming convention:

```
arch[-vendor][-os]-abi
```

On abi

```
eabi: runs on baremetal ARM ,
gnueabi: code for linux is compiled
gnueabihf: same as gnueabi with hard float
```

Examples: Native and 32-bit ARM. For 64-bit ARM see p.7.

```
$ sudo apt install gcc binutils
$ sudo apt install gcc-arm-linux-gnueabihf \
    binutils-arm-linux-gnueabihf
```

Chapter 1 - Introduction to Kernel Development
Kernel Source

- Old naming convention (until 2003): odd(unstable)-even(stable) versioning
- Semantic versioning $(\leq 2.6) : X(major).Y(minor).Z(patch) - incrementbackward - compatibleEndofsemanticversioning(3.0in2011) : Linusbumped2.6.39to3.0$
- Arbitrary versioning (3.20): Linus decided to increment X whenever Y got too big. Hence the bump from 3.20 to 4.0.
- The kernel currenly uses the X.Y versioning scheme which has nothing to do with the semantic scheme.

Chapter 1 - Introduction to Kernel Development
Kernel Release Model

There are two latest releases. Bug fixes and new features are prepared by subsystem maintainers and then Linus Torvalds merges them into mainline - his Linux tree aka master git repo.

- Stable Release - community submits through release candidate tags, Linus approves and makes final release. No strict timeline but generally mainline kernels are released every 2-3 months. When kernel is released (e.g., 4.9) the number is based on the numbering scheme used in the bugfix kernel releases (e.g., 4.9.y), which refer to a branch in the stable kernel release tree.
- Long-term support (LTS)

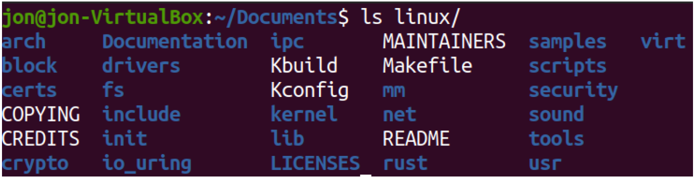
Chapter 1 - Introduction to Kernel Development
Downloading the Kernel source

We'll be using Linus' tree

```
$ git clone
    https://github.com/torvalds/linux.git —depth 1
```

Chapter 1 - Introduction to Kernel Development
Familiarize with the Kernel Source Structure (master)

List your Linux kernel source



```
jon@jon-VirtualBox:~/Documents$ ls linux/
arch     Documentation  ipc       MAINTAINERS  samples   virt
block    drivers        Kbuild    Makefile     scripts
certs    fs             Kconfig   mm           security
COPYING  include        kernel    net          sound
CREDITS  init           lib       README       tools
crypto   io_uring       LICENSES  rust         usr
```

Chapter 1 - Introduction to Kernel Development
Configuring and building the Linux Kernel

- The Kernel's Makefile invokes $(CROSS_COMPILE)gcc
- Typical configuration and build commands look like
  - ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make [menuconfig/target]
- make menuconfig will prompt a menu
  - Configuration choices: boolean, string, tristate, hex, int
- Configure the kernel to include symbols and time stamps (see book for the full list)
  - CONFIG_KALLSYMS
  - CONFIG_PRINTK_TIME

Chapter 1 - Introduction to Kernel Development
Building the Linux Kernel

Linux is a Makefile-base project. By default the make target is all

- For x86: vmlinux bzImage modules
- ARM or aarch64: vmlinux, zImage modules dtbs

bzImage and zImage are compressed kernel images. vmlinux produces a raw image and dtbs is the device tree blob binary.

```
$ ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make
```

You can build it in parallel (e.g., make -j$(nproc)). The author of the book uses nproc*2 in his build setup.

Before installing the modules make sure to add an installation directory

- ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=DIR make modules_install