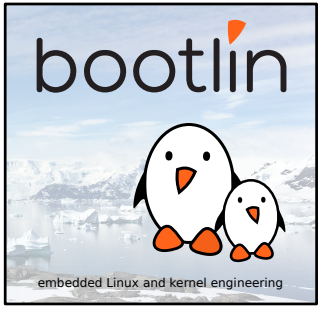
Embedded Linux - LXE22109 training

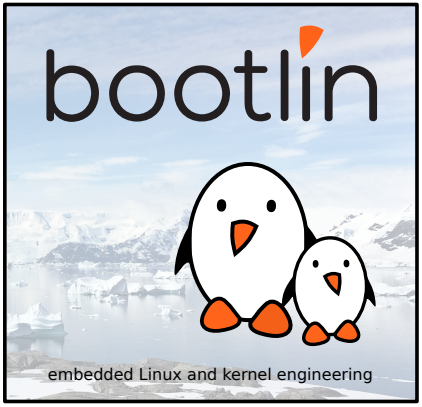


Latest update: August 24, 2022.

Document updates and training details:
https://bootlin.com/training/

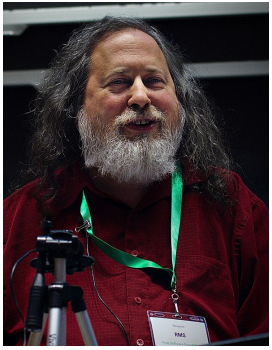
Corrections, suggestions, contributions and translations are welcome!
Send them to feedback@bootlin.com

Introduction to Embedded Linux



Birth of Free Software

- 1983, Richard Stallman, **GNU project** and the **free software** concept. Beginning of the development of *gcc*, *gdb*, *glibc* and other important tools
- 1991, Linus Torvalds, **Linux kernel project**, a UNIX-like operating system kernel. Together with GNU software and many other open-source components: a completely free operating system, GNU/Linux
- 1995, Linux is more and more popular on server systems
- 2000, Linux is more and more popular on **embedded systems**
- 2008, Linux is more and more popular on mobile devices and phones
- 2012, Linux is available on cheap, extensible hardware: Raspberry Pi, BeagleBone Black



Richard Stallman in 2019
Image credits (Wikipedia):
https://frama.link/qC73jkk4

Free software?

- A program is considered **free** when its license offers to all its users the following **four** freedoms
  - Freedom to run the software for any purpose
  - Freedom to study the software and to change it
  - Freedom to redistribute copies
  - Freedom to distribute copies of modified versions
- These freedoms are granted for both commercial and non-commercial use
- They imply the availability of source code, software can be modified and distributed to customers
- **Good match for embedded systems!**

What is embedded Linux?

Embedded Linux is the usage of the **Linux kernel** and various **open-source** components in embedded systems

Advantages of Linux and Open-Source in embedded systems

- **Ability to reuse components**
  Many features, protocols and hardware are supported. Allows to focus on the added value of your product.
- **Low cost**
  No per-unit royalties. Development tools free too. But of course deploying Linux costs time and effort.
- **Full control**
  You decide when to update components in your system. No vendor lock-in. This secures your investment.
- **Easy testing of new features**
  No need to negotiate with third-party vendors. Just explore new solutions released by the community.
- **Quality**
  Your system is built on high-quality foundations (kernel, compiler, C-library, base utilities...). Many Open-Source applications have good quality too.
- **Community support**
  Can get very good support from the community if you approach it with a constructive attitude.
- **Participation in community work**
  Possibility to collaborate with peers and get opportunities beyond corporate barriers.

A few examples of embedded systems running Linux

Wireless routers

Image credits: Evan Amos (https://bit.ly/2JzDIkv)

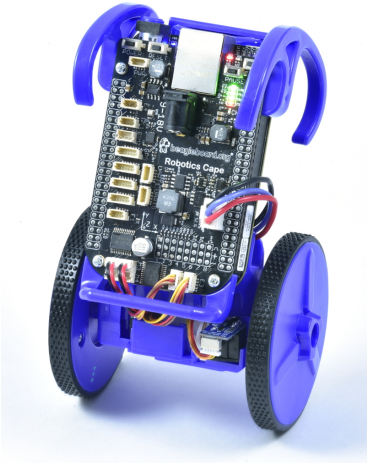
Video systems

Image credits: https://bit.ly/2HbwyVq

Bike computers

Product from BLOKS (http://bloks.de). Permission to use this picture only in this document, in updates and in translations.

Robots



eduMIP robot (https://www.ucsdrobotics.org/edumip)

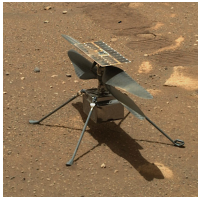
In space

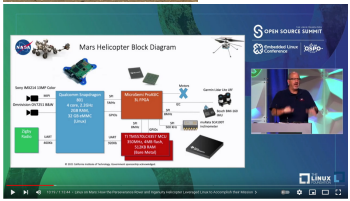
SpaceX Starlink satelites

SpaceX Falcon 9 and Falcon Heavy rockets

Image credits: Wikipedia

Mars Ingenuity Helicopter





See the *Linux on Mars: How the Perseverance Rover and Ingenuity Helicopter Leveraged Linux to Accomplish their Mission* presentation from Tim Canham (JPL, NASA): https://youtu.be/0_GfMcBmbCg?t=111

Embedded hardware for Linux systems

Processor and architecture (1)

The Linux kernel and most other architecture-dependent components support a wide range of 32 and 64 bit architectures

- x86 and x86-64, as found on PC platforms, but also embedded systems (multimedia, industrial)
- ARM, with hundreds of different *System on Chip*s (*SoC*: CPU + on-chip devices, for all sorts of products)
- RISC-V, the rising architecture with a free instruction set (from high-end cloud computing to the smallest embedded systems)
- PowerPC (mainly real-time, industrial applications)
- MIPS (mainly networking applications)
- Microblaze (Xilinx), Nios II (Altera): soft cores on FPGAs
- Others: ARC, m68k, Xtensa, SuperH...

Processor and architecture (2)

- Both MMU and no-MMU architectures are supported, even though no-MMU architectures have a few limitations.
- Linux does not support small microcontrollers (8 or 16 bit)
- Besides the toolchain, the bootloader and the kernel, all other components are generally **architecture-independent**

RAM and storage

- **RAM**: a very basic Linux system can work within 8 MB of RAM, but a more realistic system will usually require at least 32 MB of RAM. Depends on the type and size of applications.
- **Storage**: a very basic Linux system can work within 4 MB of storage, but usually more is needed.
  - **Block storage**: SD/MMC/eMMC, USB mass storage, SATA, etc,
  - **Raw flash storage** is supported too, both NAND and NOR flash, with specific filesystems
- Not necessarily interesting to be too restrictive on the amount of RAM/storage: having flexibility at this level allows to re-use as many existing components as possible.

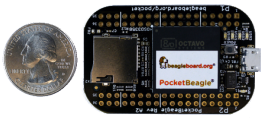
Communication

- The Linux kernel has support for many common communication buses
  - I2C
  - SPI
  - 1-wire
  - SDIO
  - PCI
  - USB
  - CAN (mainly used in automotive)
- And also extensive networking support
  - Ethernet, Wifi, Bluetooth, CAN, etc.
  - IPv4, IPv6, TCP, UDP, SCTP, DCCP, etc.
  - Firewalling, advanced routing, multicast

Types of hardware platforms (1)

- **Evaluation platforms** from the SoC vendor. Usually expensive, but many peripherals are built-in. Generally unsuitable for real products, but best for product development.
- **Component on Module**, a small board with only CPU/RAM/flash and a few other core components, with connectors to access all other peripherals. Can be used to build end products for small to medium quantities.

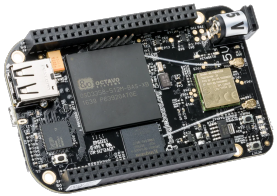
STM32MP157C-EV1 evaluation board
Image credits (st.com):
https://frama.link/NySnaxuV



PocketBeagle
Image credits (Beagleboard.org):
https://beagleboard.org/pocket

Types of hardware platforms (2)

- **Community development platforms**, to make a particular SoC popular and easily available. These are ready-to-use and low cost, but usually have fewer peripherals than evaluation platforms. To some extent, can also be used for real products.
- **Custom platform**. Schematics for evaluation boards or development platforms are more and more commonly freely available, making it easier to develop custom platforms.



Beaglebone Black Wireless board

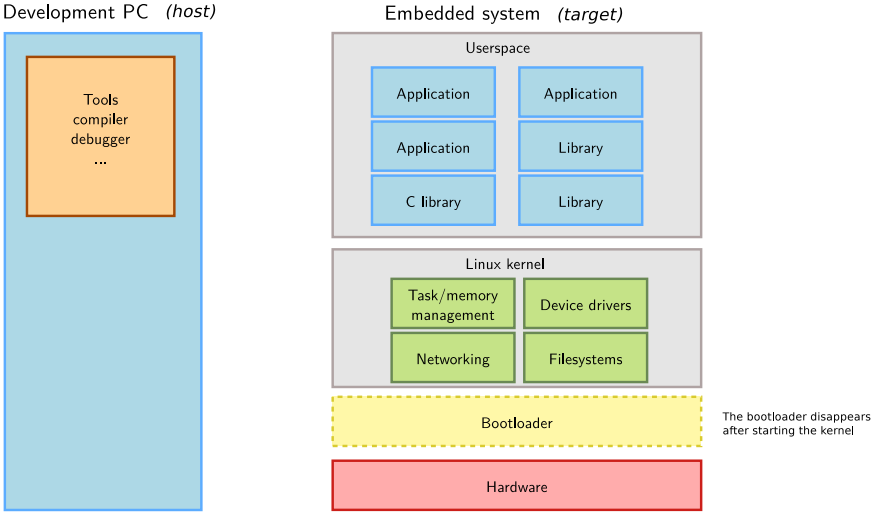
Olimex Open hardware ARM laptop main board
Image credits (Olimex): https://www.olimex.com/Products/DIY-Laptop/

Criteria for choosing the hardware

- Make sure the SoC you plan to use is already supported by the Linux kernel, and has an open-source bootloader.
- Having support in the official versions of the projects (kernel, bootloader) is a lot better: quality is better, new versions are available, and Long Term Support releases are available.
- Some SoC vendors and/or board vendors do not contribute their changes back to the mainline Linux kernel. Ask them to do so, or use another product if you can. A good measurement is to see the delta between their kernel and the official one.
- **Between properly supported hardware in the official Linux kernel and poorly-supported hardware, there will be huge differences in development time and cost.**

Embedded Linux system architecture

Host and target



Software components

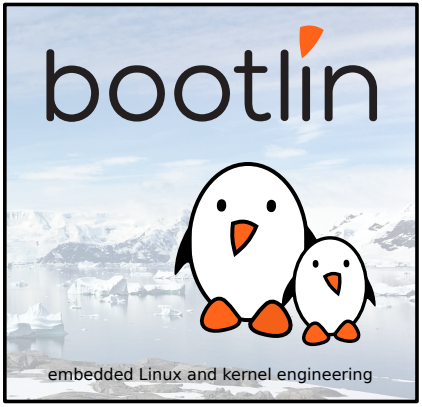
- Cross-compilation toolchain
  - Compiler that runs on the development machine, but generates code for the target
- Bootloader
  - Started by the hardware, responsible for basic initialization, loading and executing the kernel
- Linux Kernel
  - Contains the process and memory management, network stack, device drivers and provides services to user space applications
- C library
  - Of course, a library of C functions
  - Also the interface between the kernel and the user space applications
- Libraries and applications
  - Third-party or in-house

Embedded Linux work

Several distinct tasks are needed when deploying embedded Linux in a product:

- **Board Support Package development**
  - A BSP contains a bootloader and kernel with the suitable device drivers for the targeted hardware
  - Purpose of our *Kernel Development course*
- **System integration**
  - Integrate all the components, bootloader, kernel, third-party libraries and applications and in-house applications into a working system
  - Purpose of *this* course
- **Development of applications**
  - Normal Linux applications, but using specifically chosen libraries

Embedded Linux development environment



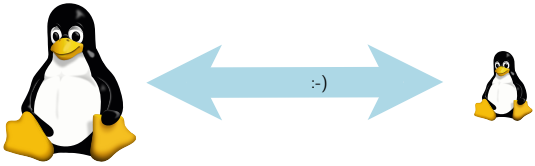
Embedded Linux solutions

- Two ways to switch to embedded Linux
  - Use **solutions provided and supported by vendors** like MontaVista, Wind River or TimeSys. These solutions come with their own development tools and environment. They use a mix of open-source components and proprietary tools.
  - Use **community solutions**. They are completely open, supported by the community.
- In Bootlin training sessions, we do not promote a particular vendor, and therefore use community solutions
  - However, knowing the concepts, switching to vendor solutions will be easy

OS for Linux development

We strongly recommend to use GNU/Linux as the desktop operating system to embedded Linux developers, for multiple reasons.

- All community tools are developed and designed to run on Linux. Trying to use them on other operating systems (Windows, Mac OS X) will lead to trouble.
- As Linux also runs on the embedded device, all the knowledge gained from using Linux on the desktop will apply similarly to the embedded device.
- If you are stuck with a Windows desktop, at least you should use GNU/Linux in a virtual machine (such as VirtualBox which is open source), though there could be a small performance penalty. With Windows 10, you can also run your favorite native Linux distro through Windows Subsystem for Linux (WSL2)



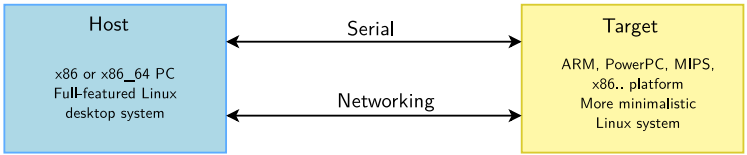
Desktop Linux distribution

- **Any good and sufficiently recent Linux desktop distribution** can be used for the development workstation
  - Ubuntu, Debian, Fedora, openSUSE, Red Hat, etc.
- We have chosen Ubuntu, derived from Debian, as it is a **widely used and easy to use** desktop Linux distribution.
- The Ubuntu setup on the training laptops has intentionally been left untouched after the normal installation process. Learning embedded Linux is also about learning the tools needed on the development workstation!

Image credits:
https://tinyurl.com/f4zxj5kw

Host vs. target

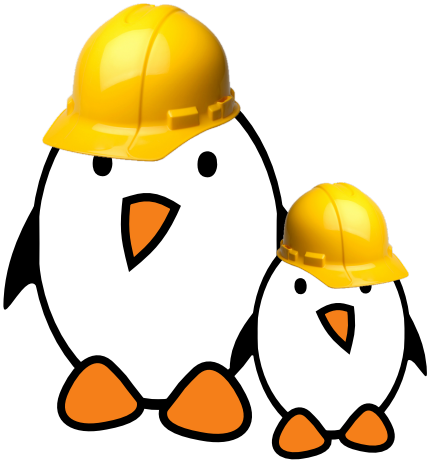
- When doing embedded development, there is always a split between
  - The *host*, the development workstation, which is typically a powerful PC
  - The *target*, which is the embedded system under development
- They are connected by various means: almost always a serial line for debugging purposes, frequently a networking connection, sometimes a JTAG interface for low-level debugging



Serial line communication program

- An essential tool for embedded development is a serial line communication program, like *HyperTerminal* in Windows.
- There are multiple options available in Linux: *Minicom*, *Picocom*, *Gtkterm*, *Putty*, *screen* and the new *tio* (https://github.com/tio/tio).
- In this training session, we recommend using the simplest of them: *Picocom*
  - Installation with `sudo apt install picocom`
  - Run with `picocom -b BAUD_RATE /dev/SERIAL_DEVICE`.
  - Exit with `[Ctrl][a] [Ctrl][x]`
- `SERIAL_DEVICE` is typically
  - `ttyUSBx` for USB to serial converters
  - `ttySx` for real serial ports
- Most frequent command: `picocom -b 115200 /dev/ttyUSB0`

Practical lab - Training Setup



Prepare your lab environment