Boot Time Optimization Training

On-line seminar

Title	**Boot Time Optimization Training**
Overview	Measuring boot time Reducing user space boot time Reducing kernel boot time Bootloader optimizations Advanced techniques and alternatives Practical demos with the ARM-based BeagleBone Black board (or with its Wireless variant).
Materials	Check that the course contents correspond to your needs: `https://bootlin.com/doc/training/boot-time`.
Duration	**Four** half days - 16 hours (4 hours per half day). 25% of lectures, 75% of practical demos.
Trainer	One of the engineers listed on `https://bootlin.com/training/trainers/`
Language	Oral lectures: English or French. Materials: English.
Audience	People developing embedded Linux systems. People supporting embedded Linux system developers.
Prerequisites	**Knowledge and practice of UNIX or GNU/Linux commands** People lacking experience on this topic should get trained by themselves, for example with our freely available on-line slides: `https://bootlin.com/blog/command-line/` **Knowledge and practice of embedded Linux system development**

Required equipment	• Computer with the operating system of your choice, with the Google Chrome or Chromium browser for videoconferencing. • Webcam and microphone (preferably from an audio headset) • High speed access to the Internet
Materials	Electronic copies of presentations, demo instructions and data.

Hardware

The hardware platform used for the practical demos of this training session is the **BeagleBone Black** board, which features:

- An ARM AM335x processor from Texas Instruments (Cortex-A8 based), 3D acceleration, etc.
- 512 MB of RAM
- 2 GB of on-board eMMC storage (4 GB in Rev C)
- USB host and device
- HDMI output
- 2 x 46 pins headers, to access UARTs, SPI buses, I2C buses and more.

Demos

The practical demos of this training session use the following hardware peripherals:

- A USB webcam
- An LCD and touchscreen cape connected to the BeagleBone Black board, to display the video captured by the webcam.
- We will also use an Arduino board as a way to measure boot time with accurary, demonstrating a hardware boot time measurement technique.

Half day 1

Lecture - Principles

- How to measure boot time
- Main ideas

Demo - Preparing the system

- Downloading bootloader, kernel and Buildroot source code
- Board setup, setting up serial communication
- Configure Buildroot and build the system
- Configure and build the U-Boot bootloader. Prepare an SD card and boot the bootloader from it.
- Configure and build the kernel. Boot the system

Lecture - Measuring time

- Generic software techniques
- Hardware techniques
- Specific solutions for each stage

Demo - Measuring time - Software solution

- Modify the system to measure time at various steps
- Timing messages on the serial console
- Timing the launching of the application

Demo - Measuring time - Hardware solution

- Measure total boot time by toggling a GPIO
- Setting up an Arduino board
- Preparing a test circuit with a 7-segment display
- Modifying the DTS to configuring Bone Black pins as GPIOs
- Making the application drive the custom GPIOs

Half day 2

Lecture - Toolchain optimizations

- Introduction to toolchains
- C libraries
- Size information
- Measuring executable performance with `time`

Demo - Toolchain optimizations

- Measuring application execution time
- Switching to a Thumb2 toolchain
- Generate a Buildroot SDK to rebuild faster

Lecture - Application optimization	**Demo - Application optimization**
• Using `strace` • Other profiling techniques	• Finding unnecessary configuration options in applications • Modifying configuration options through Buildroot • Experiments with `strace` to trace program execution

Lecture - Optimizing system initialization	**Demo - Optimizing system initialization**
• Using Bootchart • Optimizing init scripts • Possibility to start your application directly	• Using Buildroot to remove unnecessary scripts and commands • Access-time based technique to identify unused files • Simplifying BusyBox • Starting the application as the init program

Half day 3

Lecture - Filesystem optimizations	**Demo - Filesystem optimizations**
• Available filesystems, performance and boot time aspects • Making UBIFS faster • Tweaks for reducing boot time • Booting on an initramfs • Using static executables: licensing constraints	• Trying and measuring two block filesystems: ext4 and SquashFS. • Trying and measuring the initramfs solution. Constraints due to this solution.

Lecture - Kernel optimizations

- Using *Initcall debug* to generate a boot graph
- Compression and size features
- Reducing or suppressing console output
- Multiple tweaks to reduce boot time

Demo - Kernel optimizations

- Generating and analyzing a boot graph for the kernel
- Find and eliminate unnecessary kernel features
- Find the best kernel compression solution for our system

Half day 4

Demo - Kernel optimizations

Continued from the previous session

Lecture - Bootloader optimizations

- Compiling U-Boot with less features
- U-Boot configuration settings that impact boot time
- Optimizing kernel loading
- Skipping the bootloader - How to modify U-Boot to enable its *Falcon mode*

Demo - Bootloader optimizations

- Using the above techniques to make the bootloader as quick as possible.
- Switching to faster storage
- Skip the bootloader with U-Boot's *Falcon mode*

Wrap-up

- Summary of results
- Questions and answers, experience sharing with the trainer