D1Y - Embedded Linux with Yocto

Building embedded Linux platforms using Yocto

Objectives

- Understanding the architecture of the Linux system
- Learn how to install Linux on your hardware and create a BSP
- Explore the Linux system architecture
- Booting Linux
- Initializing the system
- Install existing packages on the target
- Learn how to install Linux on flash chips
- Using and customizing Yocto
- Creating Yocto-based Embedded Linux platforms
- Using Yocto to develop components

Labs can be conducted either on qemu or on target boards, that can be:
Dual Cortex/A7-based "STM32MP15-DK2" boards from STMicroelectronics.
Quad Cortex/A9-based "SabreLite" boards from NXP.
Quad Cortex/A53-based "imx8q-evk" boards from NXP.
We use the latest Yocto version supported by the chip provider
We use a recent (4.x) linux kernel, as supported by the chip supplier

Course Environment

- Theoretical course
  - PDF course material (in English) supplemented by a printed version.
  - The trainer answers trainees' questions during the training and provide technical and pedagogical assistance.
- Practical activities
  - Practical activities represent from 40% to 50% of course duration.
  - Code examples, exercises and solutions
  - One PC (Linux ou Windows) for the practical activities with, if appropriate, a target board.
    - One PC for two trainees when there are more than 6 trainees.
  - For onsite trainings:
    - An installation and test manual is provided to allow preinstallation of the needed software.
    - The trainer come with target boards if needed during the practical activities (and bring them back at the end of the course).
- Downloadable preconfigured virtual machine for post-course practical activities
- At the start of each session the trainer will interact with the trainees to ensure the course fits their expectations and correct if needed

Prerequisite

- Good C programming skills
- Knowledge of Linux user programming (see our D0 - Linux user mode programmingcourse)
- Preferably knowledge of Linux kernel and driver programming (see our D3 - Linux Driverscourse)

Target Audience

- Any embedded systems engineer or technician with the above prerequisites.

Evaluation modalities

- The prerequisites indicated above are assessed before the training by the technical supervision of the traineein his company, or by the trainee himself in the exceptional case of an individual trainee.
- Trainee progress is assessed in two different ways, depending on the course:
  - For courses lending themselves to practical exercises, the results of the exercises are checked by the trainer while, if necessary, helping trainees to carry them out by providing additional details.
  - Quizzes are offered at the end of sections that do not include practical exercises to verifythat the trainees have assimilated the points presented
- At the end of the training, each trainee receives a certificate attesting that they have successfully completed the course.
  - In the event of a problem, discovered during the course, due to a lack of prerequisites by the trainee a different or additional training is offered to them, generally to reinforce their prerequisites,in agreement with their company manager if applicable.

Plan

First Day

Introduction to Linux

- Linux history and Version management
- Linux system architecture
  - Processes and MMU
  - System calls
  - Shared libraries
- Linux system components
  - Toolchain
  - Bootloader
  - Kernel
  - Root file system
- Linux packages
- The various licenses used by Linux (GPL, LGPL, etc)

Cross compiling toolchains

- Pre-compiled toolchains
- Toolchain generation tools
  - Crosstool-ng
  - Buildroot
- Manual toolchain compilation

Linux tools for embedded systems

- Boot loaders (UBoot, Redboot, barebox)
- Optimized libraries (glibc, uClibc-ng, musl)
- Embedded GUIs
- Busybox
- Embedded distributions
  - Commercial
  - Standard
  - Tools for building custom distributions

The U-Boot bootloader

- Introduction to U-Boot
- Booting the board through U-Boot
  - Booting from NOR

    - Booting from NAND
    - Booting from eMMC
- U-Boot environment variables
    - User-defined variables
    - Predefined variables
    - Variables substitution
- The U-Boot minimal shell
    - Writing scripts in variables
    - Executing scripts
    - Using variables in scripts: the set-script pattern
- U-Boot main commands
    - Booting an OS
    - Accessing flash chips
    - Accessing file systems (NFS, FAT, EXTx, JFFS2&)
- The full U-Boot shell
    - Script structure
    - Control flow instructions (if, for&)
- Booting Linux
    - Linux kernel parameters
    - The Linux startup sequence
- Building and installing U-Boot with its native build system

Exercise: *Booting the board on NFS, using pre-existing images*
Exercise: *Configuring and building u-boot with its native build system*

Building the kernel

- The Linux build system
- Downloading stable source code
    - Getting a tarball
    - Using GIT
- Configuring the kernel
- Compiling the kernel and its modules
    - Modules delivered in-tree
    - Out-of-tree modules
- Installing the kernel and the modules

Exercise: *Configuring and compiling a target kernel for the target board with the kernel build system*

Second Day

The Linux BSP

- Linux BSP architecture
    - Overall structure
    - The ARM BSP
    - The Linux build system
- Defining and initializing the board
- Linux device drivers overview
    - Using the Flattened Device Tree

Exercise: *Create a minimal BSP for the target board*

Creating a root file system

- Packages
    - Tools to build packages (gcc, Makefile, pkg-config)
    - Autotools
    - Cross-compiling a package with autotools
- The all-in-one applications
    - Busybox, the basic utilities
    - Dropbear: encrypted communications (ssh)

- Manually building your root file system
  - Device nodes, programs and libraries
  - Configuration files (network, udev, …)
  - Installing modules
  - Looking for and installing the needed libraries
  - Testing file system consistency and completeness

Exercise: *Cross-compiling an autotools-based package*
Exercise: *Configuring and compiling Busybox and Dropbear*
Exercise: *Creating a minimal root file system using busybox and dropbear*

The Linux Boot

- Linux kernel parameters
- The Linux startup sequence
- Various initialization systems
  - busybox init
  - system V init
  - systemd
- Automatically starting an embedded system

Exercise: *Boot Linux automatically starts a user application*

Embedded file systems

- Storage interfaces
  - Block devices
  - MTD
- Flash memories and Linux MTDs
  - NOR flashes
  - NAND flashes
  - ONENAND flashes
- The various flash file system formats
  - JFFS2, YAFFS2, UBIFS
- Read-only file system
  - CRAMFS, SQUASHFS
- Standard Linux file systems
  - Ext2/3/4, FAT, NFS
- Ramdisks and initrd
  - Creating an initramfs
  - Booting through an initramfs
- Choosing the right file system formats
- Flashing the file system

Exercise: *Building an initrd root file system*

Third Day

Introduction to Yocto

- Overview of Yocto
  - History
  - Yocto, Open Embedded and Poky
  - Purpose of the Yocto project
  - The main projects
- Yocto architecture
  - Overview
  - Recipes and classes
  - Tasks

The Yocto build system

- Build system objectives
  - Building deployable images
- Layers and layer priorities
- Directory layout
- Configuration files (local, machine and distribution)
- The bitbake tool
  - Common options
- Using Yocto
  - Building a package
  - Building an image (root file system + u-boot + kernel)
- Miscellaneous tools around Yocto
  - Yocto SDK
  - Extensible SDK

Exercise: *Building a root file system using Yocto*
Exercise: *Use bitbake commands to build package & images*
Exercise: *Build an extensible SDK for the generated image*
Exercise: *Deploy the generated image using NFS*

Yocto package recipes structure

- Recipe architecture
  - Tasks
  - Task dependencies
  - Recipe dependencies
- The bitbake language
  - Standard variables and functions
  - Classes and recipes
  - The base Yocto classes
  - Main bitbake commands
- Adding a new layer
  - Layer structure
  - Various kinds of layers

Exercise: *Adding a new layer*

Fourth Day

Writing package recipes for Yocto

- Various kind of recipes and classes
  - Bare program
  - Makefile-based package
  - autotools-based package
  - u-boot
  - kernel
  - Out-of-tree module
- Recipe creation strategies
  - From scratch
  - Using devtool
  - Using recipetool
  - From an existing, similar, recipe
- Debugging recipes
  - Debugging recipe selection
  - Debugging dependencies
  - Debugging tasks
- Defining packaging
  - Package splitting
- Automatically starting a program

Exercise: *Writing a recipe for a local user-maintained package*
Exercise: *Writing and debugging a package recipe for an autotools-based package*
Exercise: *Starting a program at boot (systemd)*

Modifying recipes

- Customizing an existing package recipe (.bbappend)
- Recipe dependencies
- Creating and adding patches
  - Creating a patch for a community-provided component
  - Creating a patch for an user-maintained component
- Defining new tasks
  - Task declaration
  - Coding tasks

Exercise: *Adding patches and dependencies to a community package*
Exercise: *Adding a rootfsinstall task to directly copy the output of a user package in the rootfs image*

Fifth Day

Development process using the extensible SDK and devtool

- Using devtool to create a package and its recipe
- Using devtool to modify an existing package and recipe
- Using devtool to update a recipe to build a new version of a package

Exercise: *Create, test and modify a recipe for an existing package using devtool*

Creating new kinds of recipes

- Creating classes
  - Creating new independent classes
  - Inheriting from an existing class

Exercise: *Create a class to generalize the “rootfsinstall” task*

Creating a root file system

- Building a root file system with Yocto
  - Creating a custom root file system
- Writing an image recipe
  - Selecting the packages to build
  - Selecting the file system types
  - The various kinds of images
- Inheriting and customizing images
  - Customizing system configuration files (network, mount points, …)
- Package management
  - rpm
  - opkg

Exercise: *Writing and building an image recipe*
Exercise: *Create an image with package support for OTA deployment*
Exercise: *Test OTA update on the generated image*

Renseignements pratiques

Duration : 5 days
Cost : 2930 € HT