oD1Y - Embedded Linux using Yocto

Objectives

- Understanding the architecture of the Linux system
- Create and use a cross-development toolchain
- Learn how to install Linux on your hardware
- 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 are conducted QEMU ARM-based board
We use a recent version of Kernel
We use a recent version of Yocto

Prerequisite

- Good C programming skills (see our oL2 - C Language for Embedded MCUscourse)
- Preferably knowledge of Linux user programming (see our oD0 - Linux User Mode Programmingcourse)
- You may be interested also by the Yocto Expertcourse

Course Environment

- Theoretical course
  - PDF course material (in English).
  - Course dispensed using the Teams video-conferencing system.
  - The trainer answers trainees' questions during the training and provide technical and pedagogical assistance through the Teams video-conferencing system.
- Practical activities
  - Practical activities represent from 40% to 50% of course duration.
  - Code examples, exercises and solutions
  - One Online Linux PC per trainee for the practical activities.
  - The trainer has access to trainees' Online PCs for technical and pedagogical assistance.
  - Eclipse environment and GCC compiler.
  - QEMU Emulated board or physical board connected to the online PC (depending on the course).
  - Some Labs may be completed between sessions and are checked by the trainer on the next session.
- 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

Duration

- Total: 30 hours
- 5 sessions, 6 hours each (excluding break time)
- From 40% to 50% of training time is devoted to practical activities
- Some Labs may be completed between sessions and are checked by the trainer on the next session

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 Session

Linux overview

- Linux
  - History
  - Version management
- Linux architecture and modularity
- Linux system components
- The various licenses used by Linux (GPL, LGPL, etc)

Cross compiling toolchains

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

Exercise: *Creating a toolchain with crosstool-ng*

Linux tools for embedded systems

- Bootloaders (UBoot, Redboot, barebox)
- C libraries (glibc, eglibc, uClibc)
- Embedded GUIs
- Busybox
- Embedded distributions

The U-Boot bootloader

- Introduction to U-Boot
- Booting the board through U-Boot
  - Booting from NOR
  - Booting from NAND
  - Booting from eMMC
  - Multistage Boot
- U-Boot environment variables

  - User-defined variables
  - Predefined variables
  - Variables substitution
- The U-Boot minimal shell
- 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…)

Exercise: *Booting the board on NFS, using pre-existing images*

Second Session

Creating the embedded Linux kernel

- 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
- The Linux BSP overview
  - Structure
  - Device Drivers
  - Device Tree

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

Creating a root file system

- Packages
  - Various package build systems (autotools, CMake, …)
  - Cross-compiling a package
- 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: *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 Session

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
- 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: *Building a root file system using Yocto*
Exercise: *Build an extensible SDK for the generated image*
Exercise: *Deploy the generated image*

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 Session

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 Session

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, …)
- Users and groups management
- Package management
  - rpm
  - opkg

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

Renseignements pratiques

Duration : 30 hours
Cost : 2930 € HT