oD1Y - Linux embarqué avec Yocto

Objectifs

- Comprendre l'architecture du système Linux
- Créer et utiliser une chaîne de compilation (et développement) croisée
- Apprendre à installer Linux sur votre matériel
- Explorer l'architecture du système Linux
  - Démarrage de Linux
  - Initialiser le système
- Installer les paquets existants sur la cible
- Apprendre à installer Linux sur des puces flash
- Utilisation et personnalisation de Yocto
- Créer des plateformes Linux embarquées basées sur Yocto
- Utiliser Yocto pour développer des composants

Les travaux pratiques sont effectués sur une carte ARM QEMU
Nous utilisons une version récente de "Kernel"
Nous utilisons une version récente de Yocto

Pré-requis

- Bonnes connaissances en programmation C (voir notre cours oL2 - Langage C pour les MCUs embarqués)
- De préférence, connaissance de la programmation utilisateur Linux (voir notre cours oD0 - Programmation en mode utilisateur Linux)
- Vous pouvez également être intéressé par le cours Yocto Expert

Environnement du cours

- Cours théorique
  - Support de cours au format PDF (en anglais).
  - Cours dispensé via le système de visioconférence Teams.
  - Le formateur répond aux questions des stagiaires en direct pendant la formation et fournit une assistance technique et pédagogique.
- Activités pratiques
  - Les activités pratiques représentent de 40% à 50% de la durée du cours.
  - Elles permettent de valider ou compléter les connaissances acquises pendant le cours théorique.
  - Exemples de code, exercices et solutions.
  - Un PC Linux en ligne par stagiaire pour les activités pratiques.
  - Le formateur a accès aux PC en ligne des stagiaires pour l'assistance technique et pédagogique.
  - Certains travaux pratiques peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante.
- Une machine virtuelle préconfigurée téléchargeable pour refaire les activités pratiques après le cours
- Au début de chaque session une période est réservée à une interaction avec les stagiaires pour s'assurer que le cours répond à leurs attentes et l'adapter si nécessaire

Durée

- Totale : 30 heures
- 5 sessions de 6 heures chacune (hors temps de pause)
- De 40% à 50% du temps de formation est consacré aux activités pratiques
- Certains laboratoires peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus.

Modalités d'évaluation

- Les prérequis indiqués ci-dessus sont évalués avant la formation par l'encadrement technique du stagiaire dans son entreprise, ou par le stagiaire lui-même dans le cas exceptionnel d'un stagiaire individuel.
- Les progrès des stagiaires sont évalués de deux façons différentes, suivant le cours:
  - Pour les cours se prêtant à des exercices pratiques, les résultats des exercices sont vérifiés par le formateur, qui aide si nécessaire les stagiaires à les réaliser en apportant des précisions supplémentaires.
  - Des quizz sont proposés en fin des sections ne comportant pas d'exercices pratiques pour vérifier que les stagiaires ont assimilé les points présentés
- En fin de formation, chaque stagiaire reçoit une attestation et un certificat attestant qu'il a suivi le cours avec succès.
  - En cas de problème dû à un manque de prérequis de la part du stagiaire, constaté lors de la formation, une formation différente ou complémentaire lui est proposée, en général pour conforter ses prérequis, en accord avec son responsable en entreprise le cas échéant.

Plan

Première 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*

Deuxième 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*

Troisième 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*

Quatrième 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*

Cinquième 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

Durée : 30 heures
Prix : 2930 € HT