D1Y - Linux embarqué avec Yocto

Construire des plateformes Linux embarquées en utilisant Yocto

Objectifs

- Comprendre l'architecture du système Linux
- Apprendre à installer Linux sur votre matériel et à créer un BSP
- 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
- Utiliser et personnaliser Yocto
- Créer des plateformes Linux embarquées basées sur Yocto
- Utiliser Yocto pour développer des composants

Les travaux pratiques peuvent être menés soit sur qemu soit sur des cartes cibles, qui peuvent être :
Cartes "STM32MP15-DK2" à double Cortex/A7 de STMicroelectronics
Cartes "SabreLite" à base de Quad Cortex/A9 de NXP
Cartes "imx8q-evk" à base de Quad Cortex/A53 de NXP
Nous utilisons la dernière version de Yocto supportée par le fournisseur de la puce
Nous utilisons un noyau linux récent (4.x), tel que supporté par le fournisseur de la puce

Environnement du cours

- Cours théorique
  - Support de cours imprimé et au format PDF (en anglais).
  - 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 ou Windows) par binôme de stagiaires (si plus de 6 stagiaires) pour les activités pratiques avec, si approprié, une carte cible embarquée.
  - Le formateur accède aux PC des stagiaires pour l'assistance technique et pédagogique.
- Une machine virtuelle préconfigurée téléchargeable pour refaire les activités pratiques après le cours
- Au début de chaque demi-journée 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

Pré-requis

- Bonnes connaissances en programmation C
- Connaissance de la programmation utilisateur Linux (voir notre cours D0 - Programmation en mode utilisateur Linux)
- De préférence, connaissance de la programmation du noyau et des pilotes Linux (voir notre cours D3 - Drivers Linux)

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

Premier jour

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*

Deuxième jour

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*

Troisième jour

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*

Quatrième jour

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 jour

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

Renseignements : 5 jours