Linux embarqué avec Yocto Project

Christophe BLAESS - janvier 2020

Ce cours en ligne est prévu pour vous guider à la découverte de l'outil **Yocto Project**, permettant de produire des systèmes embarqués dont la configuration et le contenu sont parfaitement maîtrisés.

La progression est architecturée en **quatre grandes parties** :

- Produire une image avec Yocto
- Configurer le contenu de l'image produite par Yocto
- Développer du code métier pour une image Yocto
- Ajuster l'image Yocto pour une cible personnalisée

Chaque partie est elle-même découpée en plusieurs séquences, traitant d'un thème particulier. Je vous encourage à parcourir le sommaire ci-dessous pour vous familiariser avec la structure de ce cours et comprendre la progression proposée.

I – Produire une image avec Yocto

I.1 – Vocabulaire et concepts : Yocto utilise une terminologie (*layers*, *bitbake*, *recipes*, *distro*...) qui peuvent dérouter le lecteur. Cette première étape présentera rapidement les concepts spécifiques.

- Image et contenu
- Bitbake et recettes
- Layers et distributions

I.2 – Production d'une image standard : Nous préparerons ensemble une image pour une émulation de cible x86 et la testerons. L'utilisation d'un émulateur est moins excitante qu'une véritable carte embarquée, mais cette étape est importante pour valider le fonctionnement de notre plateforme de compilation en s'affranchissant des aléas de l'installation et du test de l'image sur une vraie carte.

- Préparation de l'emplacement de travail

- Poky
- Initialisation de l’environnement de travail
- Production d’une image
- Test de l’image produite
- Conclusion

I.3 – Production d’images pour des cibles spécifiques : Après l’émulateur x86, nous créerons des images pour émulation Arm, et pour deux cartes réelles : la *Beagle Bone Black* et le *Raspberry Pi*.

- Variable d’environnement MACHINE
- Image pour émulateur ARM
- Et sur une vraie cible embarquée ?
- Un Raspberry Pi, sinon rien !
- Conclusion

II – Configurer une image produite par Yocto

II.1 – Personnalisation de la configuration : nous éditerons notre fichier `local.conf` pour ajuster les répertoires de stockage temporaires, pour ajouter des utilisateurs et configurer leurs mots de passe et pour personnaliser le nom d’hôte de la machine.

- Fichier local.conf
- Utilisateurs et mots de passe
- Hostname
- Conclusion

II.2 – Ajouts d’applications dans l’image : nous ajouterons tout d’abord des utilitaires dont les recettes sont présentes dans Poky, puis des packages se trouvant dans des *layers* externes à télécharger. Après avoir examiné la notion de *features* nous créerons notre propre image pour personnaliser notre système plutôt que de surcharger le fichier `conf/local.conf`.

- Utilitaires présents dans Poky
- Ajout de packages hors Poky
- Utilisation des fonctionnalités d’image
- Création d’une image spécifique
- Conclusion

II.3 – Personnalisation des recettes : grâce aux extensions de recettes nous verrons comment surcharger un fichier d’une recette de Poky, puis comment modifier avec un *patch* un fichier provenant d’une recette et même quelques lignes d’un *package* dans une recette extérieure à Poky.

- Adaptation d’une recette de Poky
- Ajout d’un patch dans une recette
- Patch sur un fichier source de package
- Utilisation de `recipetool`
- Conclusion

III – Développer du code métier pour une image Yocto

III.1 – Ajouter des scripts ou des fichiers de données : nous allons écrire une première recette qui permettra d’installer sur la cible des fichiers préexistants comme des scripts *shells* ou Python.

- Système de fichiers en lecture seule
- Intégration de scripts shell
- Scripts et modules Python
- Conclusion

III.2 – Compiler une application métier en dehors de Yocto : deux approches sont possibles pour compiler un code métier pour une cible embarquée. Cette séquence développe une utilisation “manuelle” des outils fournis par Yocto pour produire le code métier indépendamment du système de base.

- Extraction de la toolchain
- Compilation manuelle d’un fichier source C
- Compilation avec un Makefile
- Compilation avec les autotools
- Compilation avec CMake
- Conclusion

III.3 – Intégrer notre application métier dans la production de Yocto : Une seconde approche consiste à intégrer notre code métier dans la liste des *packages* compilés par Yocto. C’est ce que nous ferons dans cette séquence où nous rédigerons plusieurs recettes.

- Recette utilisant les autotools
- Recette utilisant CMake
- Recette pour projet avec Makefile personnalisé
- Recette pour projet sans Makefile
- Conclusion

IV – Ajuster l'image Yocto pour une cible personnalisée

IV.1 – Configuration du réseau : la configuration par défaut pour le réseau est très bien adaptée la plupart du temps car elle utilise le protocole DHCP pour initialiser les interfaces disponibles. Toutefois dans certains cas, on préfère une configuration statique des paramètres réseau, c'est que que nous allons examiner.

- Configuration de référence
- Configuration initiale du réseau
- Configuration réseau statique
- Recherche de la recette responsable d'un fichier
- Conclusion

IV.2 – Configuration des utilitaires système : le Raspberry Pi ne pouvant pas maintenir l'heure lorsqu'il est éteint, il n'a aucune référence temporelle au démarrage. Nous allons lui ajouter un service NTP (*Network Time Protocol*) en modifiant la configuration de Busybox et en préparant un script de démarrage automatique au *boot*.

- Gestion de l'heure système
- Configuration de Busybox
- Démarrage d'un service
- Conclusion

IV.3 – Personnaliser le support du matériel : dans un projet de système embarqué, il est souvent nécessaire de gérer des périphériques spécifiques absents du kit de développement initial. Cela nécessite de configurer le *kernel* et de renseigner le *device tree*. Cette dernière séquence permettra d'aborder ces points techniquement plutôt avancés.

- Configuration du noyau
- Modification du *Device Tree*
- Patch sur les sources du noyau
- Conclusion

Enfin, pour ceux qui préfèrent être accompagnés pour la découverte et la maîtrise du système Yocto, j’anime des **sessions de formation** consacrées à ce sujet chez Logilin.

Ce document est placé sous licence *Creative Common CC-by-nc*. Vous pouvez copier son contenu et le réemployer à votre gré pour une utilisation non-commerciale. Vous devez en outre mentionner sa provenance.