

SOMMAIRE

1. Fondamentaux de Yocto

- Introduction aux composants clés de Yocto.
- Aperçu de l'architecture et des outils de Yocto.

2. Générer une Image de Base

- Travaux Pratiques: Utilisation de Poky pour générer une première image simple pour une carte cible spécifique (par exemple, Raspberry Pi).
- Étude de cas: Analyse des layers et des recettes impliquées dans la construction de l'image.

3. Personnalisation d'une Image Linux Embarquée

- Création et gestion de layers personnalisés.
- Ajout et suppression de packages dans une image.
- Travaux Pratiques: Personnalisation d'une image pour intégrer des applications et des services spécifiques.
- Étude de cas: Modification du noyau Linux pour ajouter un pilote de périphérique.

SOMMAIRE

4. Développement d'Applications et Utilisation du SDK

- Introduction à l'Application Development Toolkit (ADT) de Yocto.
- Création d'un environnement de développement pour applications embarquées.
- ini-Projet - Développement d'Application
- Travaux Pratiques: Développement d'une application simple utilisant le SDK Yocto, destinée à être déployée sur l'image personnalisée créée précédemment.
- Étude de cas: Intégration continue avec Yocto et tests automatisés pour le développement embarqué.

5. Introduction aux Systèmes d'Exploitation Temps Réel (RTOS)

- Brève introduction aux concepts des RTOS
- Définition et caractéristiques des RTOS
- Différences entre RTOS et systèmes d'exploitation classiques

6. Architecture et composants de FreeRTOS

- Utilisation de FreeRTOS dans les systèmes embarqués
- Intégration de FreeRTOS avec Yocto
- Création et configuration de projets FreeRTOS
- Intégration de FreeRTOS dans une build Yocto

Fondamentaux de Yocto

Introduction aux composants clés de Yocto.

Yocto est un projet open-source qui permet de créer des systèmes Linux personnalisés pour des appareils embarqués.

1. **Poky** : C'est la distribution de référence de Yocto. Poky est un modèle de départ qui inclut le script `oe-init-build-env`, l'ensemble de métadonnées de base, un ensemble de configurations prédéfinies, ainsi que BitBake, l'outil de build central de Yocto.
2. **BitBake** : C'est l'outil de build principal utilisé par Yocto. Il sert à exécuter les tâches telles que le téléchargement des sources, leur configuration, la compilation et l'assemblage des paquets logiciels. BitBake utilise des recettes et des classes pour gérer les dépendances et orchestrer le processus de construction.
3. **Recettes** : Les recettes sont des fichiers qui fournissent les instructions nécessaires à BitBake pour construire un paquet spécifique. Elles contiennent des métadonnées sur les sources du paquet, les dépendances, les instructions de compilation et d'installation, etc.
4. **Layers** : Les layers sont des collections de recettes et de classes qui permettent de modulariser et de réutiliser les configurations. Par exemple, un layer peut contenir tout le nécessaire pour supporter un type particulier de matériel ou une pile logicielle spécifique.

Fondamentaux de Yocto

Introduction aux composants clés de Yocto.

5. **Meta-Layers** : Ce sont des layers qui contiennent des informations sur d'autres layers. Par exemple, `meta-oe` est un layer qui contient des recettes supplémentaires pour des paquets couramment utilisés dans des systèmes embarqués.
6. **Toolchain** : Yocto permet de générer des toolchains croisées, qui sont des ensembles d'outils de compilation permettant de construire du code pour une architecture matérielle cible à partir d'une architecture hôte différente.
7. **Image Builder** : C'est un outil utilisé pour créer les images système finales à partir des paquets compilés. Cela inclut les systèmes de fichiers pour divers formats de périphériques et médias de stockage.
8. **SDK** (Software Development Kit) : Yocto peut générer un SDK spécifique au projet, permettant aux développeurs de créer des applications pour le système embarqué spécifique qu'ils ont construit.
9. **OE-Core** : C'est le cœur des données de base de OpenEmbedded, une partie importante de l'écosystème Yocto qui fournit les éléments essentiels nécessaires à la création d'un système d'exploitation Linux.

Fondamentaux de Yocto

Aperçu de l'architecture et des outils de Yocto

L'architecture de Yocto est conçue pour être modulaire et extensible, ce qui permet de créer des systèmes embarqués personnalisés en utilisant une série de composants interconnectés.

1. BitBake :

- Cœur du système Yocto, c'est un moteur de tâches qui interprète les recettes et orchestre le processus de construction.

2. Métadonnées :

- **Recettes** (`*.bb`) : Fichiers qui contiennent les instructions pour construire un paquet spécifique, y compris les sources, dépendances, et étapes de compilation.
- **Classes** (`*.bbclass`) : Fichiers réutilisables qui fournissent des fonctions partagées entre les recettes.
- **Configurations** (`*.conf`) : Fichiers de configuration pour les builds, y compris la configuration de la machine cible, les préférences de l'utilisateur, et les variables globales.

Fondamentaux de Yocto

Aperçu de l'architecture et des outils de Yocto

3. Layers :

- Collections organisées de recettes, classes, et configurations spécifiques à un domaine (par exemple, support matériel, interfaces utilisateur graphiques, etc.).

4. OpenEmbedded Core (OE-Core) :

- Ensemble de recettes, classes, et configurations de base qui fournissent les fonctionnalités essentielles pour un système Linux minimal.

Fondamentaux de Yocto

Aperçu de l'architecture et des outils de Yocto

- **build/** :
 - **conf/** : Contient les fichiers de configuration principaux comme **local.conf** et **bblayers.conf**.
 - **tmp/** : Le dossier de travail principal pour les builds, subdivisé en plusieurs sous-dossiers :
 - **deploy/** : Où les images finales et les paquets sont stockés après la construction.
 - **work/** : Contient les répertoires de travail pour chaque recette construite, organisés par architecture et nom de paquet.
 - **cache/** : Stocke les données de cache qui aident à accélérer le processus de build en réutilisant les résultats antérieurs.
 - **sysroots/** : Contient les systèmes racine pour chaque architecture cible et pour l'hôte, utilisés pour la compilation croisée.
 - **stamps/** : Dossiers contenant des "timestamps" indiquant quand les tâches spécifiques ont été complétées, utilisés pour gérer les dépendances entre tâches.

Fondamentaux de Yocto

Aperçu de l'architecture et des outils de Yocto

- **sources/** :
 - **meta/** : Layer de base contenant des recettes essentielles et des classes.
 - **meta-yocto/** : Contient des recettes spécifiques au projet Yocto, telles que les recettes pour l'image de démarrage de Poky.
 - **meta-yocto-bsp/** : Layer pour les BSPs.
 - **meta-<nom>** : D'autres layers pour des fonctionnalités ou des plateformes spécifiques.
 - **poky/** : Parfois présent, surtout si Poky est utilisé comme référence. Poky inclut BitBake, la documentation, les scripts de démarrage, et les métadonnées de base.
- **downloads/** : Un dossier partagé pour stocker les sources téléchargées de tous les paquets.
- **sstate-cache/** : Stocke les composants pré-compilés pour accélérer les builds répétitifs.
- **cache/** : Parfois utilisé pour conserver des données de configuration et de parsing de BitBake pour améliorer les performances des builds successifs.

Fondamentaux de Yocto

Aperçu de l'architecture et des outils de Yocto

- **scripts/** :
 - Contient des scripts utilisés pour des tâches telles que l'initialisation de l'environnement de build, l'analyse des logs, et la gestion des layers.
- **oe-init-build-env** : Script pour initialiser l'environnement de build. Il configure les variables d'environnement nécessaires pour utiliser BitBake et prépare le dossier **build/**.
- **bitbake-layers** : Utilitaire pour aider à gérer les layers dans **bblayers.conf**.
- **local.conf** et **bblayers.conf** : Comme déjà mentionné, ce sont des fichiers clés pour la configuration du processus de build, spécifiant les layers à utiliser, les paramètres de la machine cible, les options de compilation, etc.

Travaux Pratiques

1. Installer les paquets nécessaires :

- Pour Ubuntu :

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa \
libssl1.2-dev pylint3 xterm
```

2. Cloner le dépôt Poky :

- Clonez le dépôt Poky depuis GitHub :

```
git clone git://git.yoctoproject.org/poky
cd poky
```

3. Initialiser l'environnement :

- Sourcez le script d'initialisation pour configurer l'environnement :

```
source oe-init-build-env
```

Travaux Pratiques

4. Cloner les layers nécessaires :

- Vous devez ajouter le BSP pour Raspberry Pi. Clonez `meta-raspberrypi` dans le dossier `sources/` :

```
git clone git://git.yoctoproject.org/meta-raspberrypi sources/meta-raspberrypi
```

5. Configurer `bblayers.conf` :

- Éditez le fichier `conf/bblayers.conf` pour inclure `meta-raspberrypi` :

```
BBLAYERS ?= " \  
    /chemin/vers/poky/meta \  
    /chemin/vers/poky/meta-poky \  
    /chemin/vers/poky/meta-yocto-bsp \  
    /chemin/vers/poky/meta-raspberrypi \  
"
```

Travaux Pratiques

6. Modifier local.conf :

- Éditez le fichier `conf/local.conf` :
- Définissez la machine cible :

```
MACHINE ??= "raspberrypi3"
```

7. Construire l'image :

- Construisez une image simple, telle que `core-image-minimal` :

```
bitbake core-image-minimal
```