SOMMAIRE

SOMMAIRE

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 \
libsdl1.2-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
```

Création et gestion de layers personnalisés.

1. **Initialiser l'Environnement de Construction** :

```
source oe-init-build-env /home/yocto/poky/build
```

2. **Créer un Nouveau Layer** :

```
bitbake-layers create-layer meta-custom
```

3. **Ajouter le Layer à bblayers.conf** :

- Ajoutez le nouveau layer à `bblayers.conf`.

```
bitbake-layers add-layer meta-custom
```

4. **Activer/Désactiver un Layer** :
   - Pour ajouter un layer :

```
bitbake-layers add-layer <layer-path>
```

   - Pour supprimer un layer :

```
bitbake-layers remove-layer <layer-path>
```

Création et gestion de layers personnalisés - Ajout de Packages.

1. **Modifier la Recette d'Image** :
   - Ouvrez la recette de l'image que vous souhaitez modifier, par exemple `core-image-minimal.bb`.

```
nano meta-custom/recipes-core/images/core-image-custom.bb
```

2. **Ajouter les Packages** :
   - Ajoutez les packages que vous souhaitez inclure dans l'image.

```
require recipes-core/images/core-image-minimal.bb
IMAGE_INSTALL += "package1 package2"
```

3. **Supprimer les Packages** :
   - Utilisez `IMAGE_INSTALL_remove` pour supprimer les packages.

```
require recipes-core/images/core-image-minimal.bb
IMAGE_INSTALL_remove = "package-to-remove"
```

Types de Fichiers BitBake

1. **.bb (BitBake Recipe Files)**
   - **Description** : Les fichiers `.bb` contiennent des recettes BitBake. Une recette décrit comment obtenir, configurer, compiler et installer un paquet logiciel.
   - **Utilisation** : Utilisé pour créer des paquets spécifiques.
2. **.bbappend (BitBake Append Files)**
   - **Description** : Les fichiers `.bbappend` permettent d'ajouter ou de modifier des informations dans des recettes `.bb` existantes sans les dupliquer.
   - **Utilisation** : Utilisé pour ajouter des patches, modifier des variables, ou ajouter des étapes à une recette existante.
3. **.bbclass (BitBake Class Files)**
   - **Description** : Les fichiers `.bbclass` contiennent des classes BitBake. Une classe est un ensemble de fonctions et de variables qui peuvent être réutilisées dans plusieurs recettes.
   - **Utilisation** : Utilisé pour partager des comportements communs entre plusieurs recettes.

Types de Fichiers BitBake

4. **.inc (Include Files)**
   - **Description** : Les fichiers `.inc` sont des fichiers d'inclusion qui permettent de partager des fragments de recettes entre plusieurs recettes `.bb`.
   - **Utilisation** : Utilisé pour éviter la duplication de code en incluant des variables et des fonctions communes.
5. **.conf (Configuration Files)**
   - **Description** : Les fichiers `.conf` contiennent des configurations globales ou spécifiques à une build.
   - **Utilisation** : Utilisé pour configurer l'environnement de build, les machines cibles, et d'autres paramètres globaux.
   - **Exemple** : `local.conf`, `bblayers.conf`

Quand Utiliser Chaque Type de Fichier

- **.bb** : Utilisé pour définir des recettes individuelles pour des paquets logiciels spécifiques.
- **.bbappend** : Utilisé pour modifier ou étendre des recettes `.bb` existantes, particulièrement utile pour appliquer des patches ou des modifications spécifiques sans dupliquer toute la recette.
- **.bbclass** : Utilisé pour encapsuler des comportements réutilisables qui peuvent être inclus dans plusieurs recettes.
- **.inc** : Utilisé pour partager des fragments de code entre plusieurs recettes, évitant ainsi la duplication de code.
- **.conf** : Utilisé pour configurer l'environnement de build, incluant des configurations globales, des machines cibles, et des couches (layers).

Cycle de vie d'un fichier .bb

1. fetch (do_fetch) : Télécharger les sources.
2. unpack (do_unpack) : Décompresser les sources.
3. patch (do_patch) : Appliquer des patches.
4. configure (do_configure) : Configurer les sources pour la compilation.
5. compile (do_compile) : Compiler les sources.
6. install (do_install) : Installer les fichiers compilés dans le répertoire de destination.
7. package (do_package) : Créer les packages binaires.

TP 1

Objectif : Créer un custom layer nommé `meta-networking` et y ajouter une recette pour une application réseau `net-app` en utilisant toutes les étapes du cycle de vie d'une recette Yocto (fetch, unpack, patch, configure, compile, install, package).
Description de l'Application : Vous allez développer une application réseau simple en C appelée `net-app` qui écoute sur le port 8080 et renvoie un message prédéfini aux clients qui se connectent. Cette application sera intégrée dans un environnement Yocto. Vous devrez créer un custom layer, développer une recette Yocto pour cette application, et suivre toutes les étapes du cycle de vie d'une recette Yocto.

Partie 1 : Créer et Intégrer l'Application Réseau

1. **Préparer les Fichiers Sources** :
2. **Créer la Structure du Custom Layer** :
3. **Ajouter la Recette de l'Application** :
4. **Ajouter le Custom Layer au Build** :
5. **Construire l'Image** :

TP 1

Partie 2 : Appliquer un Patch à l'Application

1. **Créer un Patch pour `net-app.c`** :
2. **Ajouter le Patch à la Recette** :
3. **Reconstruire l'Image** :

TP 2

Vous allez modifier le noyau Linux utilisé par Yocto pour ajouter un pilote de périphérique. Pour cela, vous devrez créer un custom layer, développer une recette Yocto pour appliquer un patch au noyau Linux.