TP1 MI11 Linux embarqué

Semestre printemps 2022

Guillaume Sanahuja – guillaume.sanahuja@hds.utc.fr

Table des matières

Préambule

Les TPs Linux Embarqué et Xenomai s'effectuent sur une machine virtuelle à télécharger (voir instructions sur le moodle), attention l'archive fait 10.6Go et 24.4Go une fois décompressée. Le couple login/mot de passe est mi11p22/mi11p22.

Un compte rendu de TP au format PDF est à rendre sur le moodle de l'UV. Il vous est demandé d'y écrire les réponses aux questions figurant dans les différents exercices. Également, vous pouvez compléter le compte rendu avec toute information que vous jugerez utile, par exemple les manipulations que vous avez réalisées pendant la séance, les résultats que vous avez observés, etc.

Travail préalable à la séance de TP

Relire le cours sur Linux embarqué disponible sur le moodle. Prendre connaissance de la *datasheet* du Joy-Pi-Note disponible sur le moodle, et des caractéristiques de la carte RaspberryPi 4.

Question 0.1 : Décrivez les caractéristiques de la carte . Quelle est la référence du processeur de la carte, quelle est sa fréquence et son architecture ? Quelle est la quantité de mémoire vive sur la carte ?

Question 0.2 : Quelles sont les différentes possibilités pour stocker le noyau et le système de fichiers nécessaires pour démarrer la carte ?

Exercice 1 : Prise en main de l'environnement Yocto

Dans cet exercice, nous allons prendre en main l'environnement *Yocto* et recompiler l'ensemble des binaires nécessaires pour démarrer le linux embarqué sur la cible.

Les dossiers suivants sont déjà présents dans la machine virtuelle :

- */opt/mi11/poky-dunfell-23.0.16 :* sources de poky version 23, obtenues sur https://www.yoctoproject.org/
- */opt/mi11/meta-raspberrypi* : couche (layer) spécifique à la RaspberryPi pour *Yocto*
- */opt/mi11/meta-joypinote* : couche (layer) spécifique au Joy-Pi-Note pour *Yocto*
- */opt/mi11/meta-mi11* : couche (layer) spécifique à l'UV pour *Yocto*
- */opt/mi11/poky :* répertoire de compilation de *Yocto*. Afin de gagner du temps, tout a déjà été précompilé
- */opt/mi11/poky/build/conf* : répertoire de configuration de *Yocto*

Question 1.1 : Analysez brièvement le contenu des dossiers /opt/mi11/poky/build/conf, /opt/mi11/meta-raspberrypi , /opt/mi11/meta-joypinote et /opt/mi11/meta-mi11.

Afin de supporter notre cible, il faut ajouter les 3 couches spécifiques à *Yocto*. Modifiez pour cela le fichier *bblayers.conf.* Modifiez également la configuration de *Yocto* (*local.conf*) pour sélectionner la bonne cible (*joypinote*) de compilation (variable MACHINE).

Question 1.2 : Qu'avez vous changé dans les fichiers de configuration ?

Nous allons maintenant lancer la compilation du noyau et du système de fichier.

Pour initialiser *Yocto*, exécutez la commande suivante dans le terminal:

cd /opt/mi11/poky
source ../poky-dunfell-23.0.16/oe-init-build-env

Vous pouvez vérifier que les couches spécifiques ont bien été ajoutées avec la commande :

bitbake-layers show-layers

Gardez ce terminal ouvert pour les futures compilations avec *Yocto*. Pour lancer une compilation, il faut utiliser la commande *bitbake* suivie de l'image à créer. Les fichiers générés par le système de compilation seront placés dans */opt/mi11/poky/build/tmp/deploy.*
Afin de faire un système de fichiers basique, lancez :

bitbake core-image-base

Cette commande va prendre du temps mais est grandement accélérée par le fait que tout est déjà précompilé. Assurez-vous bien que dans le résumé donné par *bitbake* lors de la compilation vous avez la bonne *MACHINE*.

Question 1.3 : Analysez le résultat de la compilation se trouvant dans le répertoire /opt/mi11/poky/build/tmp/deploy/images. Quels sont les différents fichiers, à quoi servent-ils ? Quelle est la taille des fichiers générés ? Comparez avec ceux de votre VM sous Ubuntu. Pourquoi

y a t-il une différence ?

Lancez maintenant la commande suivante afin de compiler la chaîne de compilation croisée :

bitbake meta-toolchain

Ouvrez un nouveau terminal et installez la chaîne de compilation croisée avec les commandes suivantes :

cd /opt/mi11/poky/build/tmp/deploy/sdk
sudo ./poky-glibc-x86_64-meta-toolchain-cortexa7t2hf-neon-vfpv4-joypinote-toolchain-3.1.16.sh

Utilisez les réponses par défaut en appuyant sur entrée à chaque question.

Exercice 2 : Démarrer sur le noyau et le système de fichiers générés

Les cibles embarquées offrent dans la majorité des cas une interface de communication par port série disponible dès le démarrage. Le RaspberryPi est ainsi reliée au PC de développement via un convertisseur USB/série. Allumez la carte et utilisez cette interface (ttyUSB0 , 115200bps) pour recevoir les messages de la cible sur le PC. Dans le cas du RaspberryPi, des messages sont aussi visibles sur l'écran qui y est connecté.

Question 2.1 : Décrivez la séquence de démarrage. Que se passe-t-il et pourquoi (analysez la configuration sur la VM)? Quels fichiers sont manquants et où faudrait-il les mettre ? A quoi servent ces fichiers ?

Question 2.2 : Si les fichiers manquants étaient présents, que se passerait-t-il ensuite ? Quel serait le problème suivant ?

Question 2.3 : Compte tenu des questions précédentes, que faut-t-il faire pour démarrer sur le noyau et le système de fichiers que vous avez générés ? Vous devez vous adapter à la configuration des différents services, et ne pas les modifier.

Effectuez ces opérations et constatez le bon fonctionnement. Le login de la cible est *root* avec un mot de passe vide.

Question 2.4 : Décrivez le processus de boot complet de la cible. Quelles sont les applications démarrées et dans quel ordre ? Fournissez dans le rapport de TP l'ensemble des messages de sortie du terminal entre l'allumage de la cible et le prompt de login.

Pour plus de confort, connectez vous à la cible par ssh, avec la commande suivante :

ssh root@ip_cible

Où *ip_cible* est à adapter à votre situation.

Question 2.5 : Quelle est l'ip de la cible ? Comment l'avez vous trouvée ?

Affichez dans la console le contenu du fichier /proc/devices.

Question 2.6 : Que contient le fichier /proc/devices ? Quelle est la différence entre les 2 sections que l'on trouve dans ce fichier ? Identifiez le périphérique correspondant au port série que vous utilisez, mettez le en évidence dans le fichier /proc/devices et affichez la(es) ligne(s) correspondantes dans le dossier /dev. A quoi correspond le numéro que l'on trouve en début de ligne dans le fichier /proc/devices ?

Question 2.7 : Analysez (ls -l) le contenu de /bin, /sbin, /usr/bin et /usr/sbin. Qu'y a t-il de particulier ? Vous pouvez également lancer la commande busybox *sur la cible. Quel est l'avantage de busybox?*

Exercice 3 : Ajout de paquets

Le système de fichier généré est basique et contient peu de paquets. Nous allons voir comment l'étoffer. Utilisez *bitbake* pour compiler le paquet nano. Il s'agit d'un éditeur de texte.

Question 3.1 : Analysez le contenu du dossier /opt/mi11/poky/build/tmp/deploy/ipk. Comment est il organisé ? Que contiennent les sous dossiers ? Où se trouve le noyau ? Où se trouve nano?

Question 3.2 :Quels sont les différents paquets relatifs à nano?

Copiez le fichier *nano_2.2.5-r3.0_cortexa7t2hf-neon-vfpv4.ipk* dans le système de fichiers de la cible et installez le avec le gestionnaire de paquets sur la cible:

opkg install nano_2.2.5-r3.0_cortexa7t2hf-neon-vfpv4.ipk

Question 3.3 : Quel est le problème rencontré et comment pourrait-on le résoudre ?

Pour résoudre le problème de façon plus efficace, nous allons configurer opkg pour qu'il télécharge les paquets depuis un serveur. Editez le fichier */etc/opkg/opkg.conf* sur la cible et ajoutez :

src/gz cortexa7t2hf-neon-vfpv4 http://192.168.0.1/cortexa7t2hf-neon-vfpv4
src/gz joypinote http://192.168.0.1/joypinote

Mettez à jour opkg et installez de nouveau nano :

opkg update
opkg install nano

NB : si la paquet nano n'est pas trouvé par opkg, il faut au préalable forcer la reconstruction de l'index des paquets via la commande *bitbake package-index.*

Question 3.4 : Expliquez ce qu'il se passe alors. Que faut-il comme serveur sur votre VM pour que cela fonctionne ?

Exercice 4 : Compilation manuelle du noyau

L'un des buts de la seconde séance de TP sera de faire clignoter les leds (STATE_LED sur le joypinote), qui sont connectées sur des GPIOs (General Purpose Input Output). Nous devons donc nous assurer aujourd'hui que celles-ci sont opérationnelles. Les leds sont visibles dans le dossier */sys/class/leds.* Il est alors possible de contrôler l'état de la led en écrivant 0 ou 1 dans le fichier *brightness* d'une led donnée. Par exemple :

echo 1 > /sys/class/leds/nom_led/brightness

Question 4.1: Vérifiez cette fonctionnalité sur la cible.

Afin d'ajouter une fonctionnalité au noyau, nous allons le compiler manuellement, comme vu en cours. Les sources du noyau se trouvent dans le dossier : */opt/mi11/linux-raspberrypi*

Il va falloir utiliser la chaîne de compilation croisée installée précédemment. Pour cela utilisez le script fourni avec la chaîne de compilation croisée :

. /opt/poky/3.1.16/cortexa7thf-neon-vfpv4/environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-gnueabi

Question 4.2 : A quoi sert ce fichier ? Quel est le préfixe de la chaîne de compilation croisée ?

Question 4.3 : Comment obtenir la liste des configurations par défaut du noyau pour une architecture ARM ? Quelle est la configuration par défaut pour la joypinote?

Effectuez la configuration par défaut; puis lancez la personnalisation du noyau (*menuconfig*). Nous avons besoin d'y ajouter un driver pour les leds connectées par GPIO.

Question 4.4 : Quelle est l'option à activer dans le noyau ? Quelles sont les différentes possibilités pour l'activer ?

La fonctionnalité étant activée dans le noyau, il reste à définir des leds dans le fichier *device tree*. Editez le fichier *arch/arm/boot/dts/bcm2711-rpi-4-b.dts* dans les sources du noyau. Deux leds sont déjà définies dans la section *&leds* . Ces leds ne sont cependant pas visibles car la carte est enfermée dans le joypinote. Ajoutez donc la déclaration de deux nouvelles leds à coté en vous basant sur la syntaxe suivante :

```
ma_led {
        label = "ma_led";
        gpios = <&gpio num_gpio GPIO_ACTIVE_LOW>;
};
```

Les numéros de GPIOs (num_gpio) à utiliser sont les 5 et 6.

Lancez la compilation du noyau et du dtb en utilisant les cibles *bzImage* et *dtbs* du Makefile. Pensez à ajouter l'option *-j* pour lancer plusieurs compilations en parallèle et profiter de tous les threads disponibles sur votre machine. Par exemple *-j2.*

Question 4.5 : Où se trouve le résultat de la compilation ?

Copiez les fichier *zImage* et *bcm2711-rpi-4-b.dtb* pour qu'ils soient utilisés par la cible.

Question 4.6 : Comment vérifier dans les logs de démarrage que le noyau utilisé est bien celui qui vient d'être compilé ? Vérifiez également que les leds sont maintenant accessibles.

Vous pouvez alors manipuler les leds comme décrit précédemment. Il est aussi possible d’ajouter un trigger à une led. Pour cela regardez le contenu du fichier *trigger* dans le dossier de la led. Ecrivez ensuite dans ce fichier pour y associer un trigger.

Question 4.7 : Quelle commande avez vous utilisée pour ajouter un trigger ?

Exercice 5 : Compilation manuelle des modules noyau

Lors de l'exercice 4, nous ne nous sommes pas occupés des modules noyau, car ceux-ci étaient déjà présents sur le systèmes de fichiers.

Question 5.1 : Donnez la marche à suivre pour compiler les modules et les installer dans le système de fichiers.