Partie 1: Construction d'OS avec Buildroot et chaine de cross-compilation

Q1: Utilité et syntaxe

- **configs/embsys_defconfig**
  - **Utilité**

    permet de configurer des options par défaut pour une cible particulière.

  - **Syntaxe**
- **busybox.config**
- **Users.table**

Q1: Pour une cible RaspberryPi3 le ficher de configuration par défaut est :**make raspberrypi3_defconfig**

Q2: La commande **make embsys_defconfig** permet de configurer le buildroot pour utiliser le template **embsys_defconfig** puis crée un nouveau fichier .config.

Q4: le répertoire package contient l'ensemble des règles de compilation.

Q5: Utilité des différents fichiers du répertoire *package/openssh*

- ***Config.in****:* les fichiers contiennent des entrées pour presque tout ce qui est configurable dans Buildroot
- **Openssh.hash** : contient les hachages des fichiers téléchargés pour le Openssh package
- **Openssh.mk**: contient le nom du module, placé entre des séparateurs constitués de 80 hachages.
- **Sshd.service :** permet de démarer un daemon
- **S50sshd**: est un fichier.c qui exécute les services (démarrage, arrêt) d'un daemon
- **patch**: sont des fichiers correctifs neccessaire lorsqu'un processus ne démarre pas.

Q6:

- **fichier .cfg** : permet de configurer la création et la génération de différents images pour différentes architectures raspberriPI
- **Post-build.sh:** script personnalisé à exécuter avant de créer des images du système de fichiers
- **Post-image.sh:** script personnalisé à exécuter après la création des images du système de fichiers
- **Readme.txt:** fichier guide donnant un ensemble d'instructions à l'utilisateur.

Q7:

- l'architecture matérielle cible : ARM (little endian)
- le CPU ciblé : Cortex-A53
- l'ABI (en rappellant la signification de celle choisie) : EABIhf
- la librairie C utilisée : uClibc-ng
- la version du cross-compilateur : gcc 6.x
- la version du kernel: Custom Git repository rpi-4.14.y

Q8: Target packages > Networking applications > openssh => sera compilé

Q9: Busybox est un paquet qui construit une distribution Linux complète à partir des sources dépasse un peu le cadre de cette FAQ.

make busybox-menuconfig : pour modifier certains paramètres BusyBox (configuration Busybox)

aprés configurer busybox exécutez make, il devient prêt à programmer les images de noyau et de système de fichiers nouvellement compilés sur la carte et à démarrer. La programmation Flash réelle dépend de votre système, du chargeur de démarrage, du type de Flash, etc., et dépasse le cadre de cet article.

Q10:

Output/host: contient les répertoires suivants *:* arm-buildroot-linux-uclibcgnueabihf, bin, doc, etc, include, lib, libexec, man, sbin, share, usr.

Le binaire output/host/usr/bin/arm-linux-gcc correspond au: C++ compiler

Q11:

Pour **file hw** on ontient:

hw: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=b63e2cf0005a62f225b9f34534b0b2e2cbb1ef82, not stripped

Cela dit qu'il fournit une compatibilité partielle avec GNU ce qui signifie que certains fichiers binaires fonctionneront réellement.

Pour *./hw on obtient:*
Hello Worlds! (exécuté)

Q12:

Pour **file** hw on ontient:

hw: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, not stripped

Le fichier helloworld.c est compilé avec le fichier arm-linux-gcc où son processeur ont une largeur de *32 bits.*

Pour *./hw* on obtient :
bash: ./hw: cannot execute binary file: Exec format error

ça pas marché parce qu' on essaye d'exécuter un exécutable compilé pour une architecture ARM (32 bits) sur une architecture x86-64.

Partie 2: QEMU

Q1: Lorsqu'on la lance la commande --privileged, Docker va lui donner tous les droits, y compris celui de lancer de nouveaux container sur la machine hôte.

Q2: La commande chroot permet d'activer l'émulateur

Q3: le binaire cross-compilé hw dans l'environnement chroot parce qu'on exécute le fichier avec l'émulateur qui permet d'exécuter du code binaire prévu pour un certain processeur sur un autre processeur qui a des opcodes complètement différents.

Partie 3: Flashage de la carte et Bootloader

Q1: Il y'a 2 partitions sur la carte SD, dans la partition /dev/sdb1, on a le bootloader, le rule file system et dans la partition /dev/sdb2 on a l'image du Kernel

Q2: Le port TX GPIO14 est , le port RX est GPIO15

Q3: la configuration du port série permettant une communication avec la RPI3 est **ttyUSB0**

Q4: l'adresse IP de notre RPI3 est 172.20.10.229. La commande utilisé est **ifconfig**

Q5: La différence est qu'on observe qu'on peut se connecter en user mais pas en root en utilisant ssh parce que c'est le démon ssh qui tourne sur la rapsberry Pi.

Q6:

- **Setenv : sert à** modifier l'environement U-Boot
- **fatload** charger un fichier binaire à partir d'un système de fichiers dos
- **Bootz : permet de démarrer un noyau Zimage dans U-Boot**