MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Objectifs

Ce travail pratique vise les objectifs suivants :

- Mise en œuvre d'un système embarqué sous Linux
- Mise en œuvre de l'environnement de développement de systèmes embarqués sous Linux
- Debugging d'applications sous Linux embarqué
- Mise en production d'un système embarqué sous Linux

Activités

Il comprendra les activités suivantes :

1. Mise œuvre de l'environnement de développement sous Linux (machine native ou virtuelle), de préférence sous Fedora 32, ceci comprend :
    a. Installation de la machine hôte
    b. Création de l'espace de travail sur la machine hôte
    c. Génération de l'environnement de développement (toolchain, U-Boot, Linux kernel, rootfs)
2. Mise en œuvre de l'infrastructure (cible, machine hôte)
    a. Mise en place de l'infrastructure
    b. Gravure (burning) de la carte SD pour la cible
    c. Test de l'environnement de production sous carte SD
    d. Test de l'environnement de développement sous tftp/nfs
3. Debugging d'une application simple depuis la machine hôte
    a. Génération d'une application sur la machine hôte
    b. Lancement de l'application sur la cible
    c. Debugging de l'application avec Eclipse ou VS-Code et SSH
4. Mise en production
    a. Génération d'une application
    b. Génération du rootfs avec l'application
    c. Test et validation du produit

MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Informations pratiques

Ce paragraphe donne quelques informations utiles pour la réalisation de ce travail pratique.

Installation de la machine hôte

Si vous disposez déjà d'une machine Linux, vous pouvez bien naturellement déployer l'environnement de développement directement sur votre machine personnelle. Pour cela, il suffit de suivre les instructions sous option 1. Si vous préférez utiliser la machine virtuelle Fedora 32 préparée spécialement pour les travaux pratiques de ce module, veuillez simplement suivre les instructions sous option 2.

Option 1 : Création de l'espace de travail sur la machine hôte personnelle

Pour créer l'espace de travail sur la machine hôte, veuillez suivre les instructions ci-dessous.
Note : Bien que ces scripts aient été testés sur des machines vierges Ubuntu Desktop 20.04 et Fedora 32 Workstation, il est recommandé de les vérifier et de les adapter au besoin.

- Créer le répertoire « workspace » sur la « home directory »
```
$ mkdir -p ~/workspace
$ cd ~/workspace
```
- Cloner le dépôt GIT contenant les scripts d'installation de l'environnement
```
$ git clone https://gitlab.forge.hefr.ch/embsys/linuxenv.git
```
- Générer l'environnement de développement (cette phase prend passablement de temps, >1h).
```
$ ./linuxenv/nano-env
$ cd ~/workspace/nano/buildroot
$ make
$ mkdir -p ../rootfs
$ sudo tar xf output/images/rootfs.tar -C ../rootfs
```

Option 2 : Installation de la machine virtuelle Fedora 32

Pour utiliser la machine virtuelle, vous devez disposer d'un hyperviseur sur votre machine hôte. Vous devez déployer VMware (logiciel sous licence). Pour installer l'hyperviseur sur la machine hôte, veuillez suivre les instructions ci-dessous :

- Copier de la clef USB l'hyperviseur sur la machine personnelle
  - VMware-workstation-full-15…exe pour Windows
- Pour obtenir la licence VMware, ouvrir le lien : **http://hes-so.onthehub.com**
  - Se loger avec le compte HES-SO et aller sous « Students » / « VMware »
  - Choisir le logiciel correspondant à la machine personnelle
  - Ajouter le logiciel dans le panier « Add To Cart »
  - Effectuer le login Switch « HES-SO - Haute école spécialisée de Suisse occidentale »
  - Effectuer l'achat « Check Out » / « Proceed With Order »
  - Installer la licence
- Login et mot de passe
  - Username : lmi
  - Password : lmi
  - Root : lmi

MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Génération et installation de l'environnement

Si vous effectuez des modifications de la configuration du noyau ou du rootfs, il faut régénérer les différents packages et recréer les images. Pour cela, il suffit d'effectuer les commandes suivantes :

```
$ cd ~/workspace/nano/buildroot
$ make
$ sudo rm -Rf ../rootfs
$ mkdir ../rootfs
$ sudo tar xf output/images/rootfs.tar -C ../rootfs
```

Gravure de la carte SD

Lors de la génération des différents packages, Buildroot termine par la création d'une image complète de l'environnement à graver directement sur la carte SD. La génération de cette image est effectuée par l'utilitaire `"genimage"`, lequel est lancé par le script `"genimage.sh"` et sur la base du fichier de configuration `"genimage.cfg"` situé dans le répertoire de builroot `"./board/friendlyarm/nanopi-neo-plus2/"`.

Cette image `"sdcard.img"`, stockée sous le répertoire buildroot `"./output/images"`, contient :

- Directement gravé dans les secteurs de la carte SD :
  - Du 1st stage bootloader (`sunxi-spl.bin`)
  - L'U-Boot (`u-boot.itb`)
- Une partition vfat avec les fichiers utilisés par l'U-Boot
  - Le noyau Linux (`Image`)
  - Le fichier de configuration du noyau (`nanopi-neo-plus2.dtb`)
  - Le fichier de configuration de l'U-Boot pour le lancement du noyau Linux brûlé sur la carte SD (`boot.scr`)
  - Eventuellement, le fichier avec les variables d'environnement de l'U-Boot (`uboot.env`)
- Une partition ext4 avec le rootfs.

Avant de procéder à la gravure (burning) de la carte SD, il faut trouver sur quel device (`/dev/sd<x>`) les volumes sont attachés. Pour cela il suffit de taper la commande : `$ mount`

On peut ensuite créer un script bash avec les commandes nécessaires pour copier l'image sur la carte SD

- Pour détacher les volumes : `$ umount /dev/sd<x>?`
- Pour copier l'image sur la carte SD :
  ```
  $ sudo dd if=./output/images/sdcard.img of=/dev/sd<x>
  ```
- Pour fixer le nom de la 1ère partition vfat avec les fichiers de boot :
  ```
  $ sudo fatlabel /dev/sd<x>1 BOOT
  ```
- Pour fixer la taille et le nom de la 2e partition ext4 avec le rootfs :
  ```
  $ sudo e2fsck -f /dev/sd<x>2
  $ sudo resize2fs /dev/sd<x>2
  $ sudo e2label   /dev/sd<x>2 rootfs
  ```

Pour tester la bonne gravure de la carte SD, il suffira de l'installer sur la cible et de la redémarrer. On pourra voir la séquence de lancement (boot sequence) sur la console grâce à l'utilitaire `"minicom"`.

MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Mise en place de l'infrastructure réseau

Pour simplifier le développement et la réalisation des différents travaux pratiques de ce cours, il est recommandé de configurer l'ensemble de l'infrastructure réseau avec des adresses IP statiques. On utilisera de préférence, l'adresse IP 192.168.0.14 pour la cible et l'adresse IP 192.168.0.4 pour la machine hôte de développement. Cependant avant de procéder à la configuration des interfaces réseau, il faudra d'abord connecter physiquement la cible à la machine hôte. Cette connexion peut être réalisée soit par l'intermédiaire d'une interface Ethernet de la machine physique, soit par un adaptateur Ethernet/USB connecté directement à la machine virtuelle. Dans le premier cas, il faudra encore installer un bridge entre la machine physique et la machine virtuelle. Pour éviter des instabilités de connexion, il est recommandé de donner une adresse fixe à l'interface de la machine physique, p. ex. 192.168.0.3.
Pour tester le bon fonctionnement de l'installation et valider la connexion Ethernet/IP, il faut se logger sur la cible et ensuite taper les commandes ci-dessous :

```
# ifconfig eth0 192.168.0.14
# ping 192.168.0.4
```

Pour conserver la configuration de l'interface Ethernet lors de redémarrage de la cible, il faut adapter le fichier "`/etc/network/interfaces`" en ajoutant à la fin du fichier les lignes suivantes :

```
auto eth0
iface eth0 inet static
      address 192.168.0.14
      netmask 255.255.255.0
      gateway 192.168.0.4
```

Pour faciliter de debugging d'applications avec Eclipse ou VS-Code, il est également recommandé de configurer la cible avec SSH et en autorisant l'accès sans mot de passe et avec le login « root ». Par défaut, la cible tourne un server SSH « `Dropbear` ». Pour tester la connexion depuis la machine hôte, la commande ci-dessous permettra de vérifier son bon fonctionnement :

```
$ ssh root@192.168.0.14
```

Il est cependant également possible d'utiliser le serveur de « `OpenSSH` ». Dans ce cas, pour se logger sur la cible en « `root` » et sans « `password` », il faut adapter le fichier "`/etc/ssh/sshd_config`" avec vi en modifiant les lignes suivantes :

```
PermitEmptyPasswords yes
PermitRootLogin yes
```

et finalement relancer de daemon SSH en tapant la commande

```
# /etc/init.d/S50sshd restart
```

MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Mise en place de l'espace de travail sous NFS

Il est possible d'augmenter le confort de travail et l'efficacité en attachant l'espace de travail de la machine de développement hôte directement sur la cible. Ceci permet en effet d'accéder directement depuis la cible les fichiers et applications générés sur la machine hôte, évitant ainsi des copies inutiles de fichiers. Il est possible d'attacher l'espace de travail soit manuellement soit automatiquement.
Pour attacher manuellement l'espace de travail de la machine hôte sur la cible via NFS, il faut :

- Se logger sur la cible (username : « root » et password : lasser vide)
- Créer la première fois un point d'attachement sur la cible :
  `# mkdir -p /usr/workspace`
- Attacher le workspace de la machine hôte sur la cible :
  `# mount -t nfs 192.168.0.4:/home/lmi/workspace /usr/workspace -o nolock`
- Détacher le workspace de la cible :
  `# umount /usr/workspace`

Pour attacher automatiquement l'espace de travail de la machine hôte sur la cible via NFS, il suffit d'effectuer une seule fois les instructions ci-dessous :

- Créer la première fois un point d'attachement sur la cible, p. ex. :
  `# mkdir -p /usr/workspace`
- Editer le ficher "`/etc/fstab`" avec vi et ajouter la ligne ci-dessous :
  `192.168.0.4:/home/lmi/workspace /usr/workspace nfs hard,nolock 0 0`
- Activer la configuration :
  `# mount -a`

Génération d'applications sur la machine de développement hôte

Pour la génération d'applications sur la machine hôte, il est recommandé de développer ces propres Makefile. Quelques infos utiles pour trouver la toolchain et éditer le Makefile :

```
# Makefile toolchain part
TOOLCHAIN_PATH=~/workspace/nano/buildroot/output/host/usr/bin/
TOOLCHAIN=$(TOOLCHAIN_PATH)aarch64-none-linux-gnu-

# Makefile common part
CC=$(TOOLCHAIN)gcc
LD=$(TOOLCHAIN)gcc
AR=$(TOOLCHAIN)ar
CFLAGS+=-Wall -Wextra -g -c -mcpu=cortex-a53 –O0 -MD -std=gnu11
```

Debugging de l'application sur la cible

Le debugging d'application peut être effectué de deux manières, soit directement sur la cible avec `"gdb"` ou à distance en utilisant un `"gdbserver"` sur la cible et `"gdb"` sur la machine hôte. Cette dernière option offre l'avantage de pouvoir utiliser Eclipse ou VS-Code.

Eclipse

Sous Eclipse, le lanceur automatique *« GDB (DSF) Automatic Remote Debugging Launcher »*, qui utilise une connexion SSH entre la cible et la machine hôte, a le grand avantage de simplifier les opérations à effectuer pour débuter une session de debugging, « un seul clic de souris ».
Après avoir créé un projet, il faut encore configurer le débuggeur d'Eclipse en effectuant les instructions suivantes :

- Aller dans Debug Configurations et créer une nouvelle « C/C++ Remote Application »
- Choisir le lanceur automatique :
  - GDB (DSF) Automatic Remote Debugging Launcher
- Configurer « C/C++ Application » :
  - app
- Configurer une nouvelle connexion :
  - Type : SSH
  - Host : 192.168.0.14
  - User : root
- Configurer le « remote absolute file path » :
  - /usr/workspace/csel1/environment/fibonacci/app
- Configurer le non-chargement de l'application sur la cible
  - X Skip download to target path
- Configurer le débogueur GDB (Debugger / Main) :
  - /home/lmi/workspace/nano/buildroot/output/host/usr/bin/aarch64-none-linux-gnu-gdb

VS-Code

Visual Studio Code avec l'extension « C/C++ 'ms-vscode-cpptools' » offre également une interface très simple et un débuggeur de qualité pour le debugging de vos applications. Pour configurer VS-Code afin de débugger votre application, 3 opérations suffisent :

1. Ouvrir VS-Code dans le workspace où se trouve votre application.
2. Créer une tâche permettant lancer le « gdbserver » sur la cible
3. Créer un « launcher » pour le débuggeur

Une fois ces configurations terminées, il est possible de débugger votre application en effectuant les 2 opérations suivantes :

1. Lancer dans le terminal la tâche « gdbserver » (run task)
2. Lancer le débugger « gdb launch »

Voici 2 manières d'ouvrir un workspace sous VS-Code :

1. Ouvrir une shell, puis se déplacer dans le workspace et entrer « `$ code .` »
2. Dans VS-Code ouvrir le workspace avec le menu « `File -> Open Workspace` »

MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Voici un exemple de fichier de configuration de tâches (`tasks.json`), qui permet de lancer le « `gdbserver` » sur la cible et l'application « **`…/csel1/environment/fibonacci/app`** » :

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
          "label": "gdbserver",
          "type": "shell",
          "command": [
              "ssh -t root@192.168.0.14 '/usr/bin/gdbserver :1234",
              "/usr/workspace/csel1/environment/fibonacci/app 2'"]
        }
    ]
}
```

Voici un exemple de fichier de configuration (`launch.json`), qui permet de lancer le débugger « `gdb` » sur la machine hôte en lui spécifiant l'application à débugger (`"${workspaceFolder}/app"`) et de l'attacher avec le « `gdbserver` » lancé précédemment :

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "(gdb) Launch",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/app",
      "stopAtEntry": true,
      "cwd": "${workspaceFolder}",
      "externalConsole": false,
      "MIMode": "gdb",
      "miDebuggerServerAddress": "192.168.0.14:1234",
      "miDebuggerPath":
        "/home/lmi/workspace/nano/buildroot/output/host/usr/bin/aarch64-none-linux-gnu-gdb"
    }
  ]
}
```

Ces deux fichiers doivent bien évidemment être adaptés à chaque projet (path et nom de l'application).

MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Mise en place de l'environnement pour le développement du noyau sous NFS

Pour le développement de modules et de pilotes de périphériques devant fonctionner dans le noyau Linux, il est généralement plus commode de charger le noyau de la machine hôte sur la cible et d'accéder le rootfs fraichement généré ou modifié directement via le réseau.

Pour ce faire, il faut créer pour l'U-Boot un nouveau fichier de configuration, p.ex. `"boot.nfs"`, et le charger sur la carte SD dans la partition vfat BOOT. Pour créer ce fichier, il faut tout d'abord éditer un fichier avec les variables et commandes nécessaire à l'U-Boot :

- Ouvrir/créer un fichier de commandes `"boot.cmd"` et l'ouvrir à l'aide d'un éditeur de texte
- Entrer les lignes suivantes :

```
setenv ipaddr       192.168.0.14
setenv serverip     192.168.0.4
setenv netmask      255.255.255.0
setenv gatewayip    192.168.0.4
setenv board        nano
setenv hostname     myhost
setenv mountpath    /home/lmi/workspace/$board/rootfs
setenv tftppath     /$board/buildroot/output/images
setenv bootargs     console=ttyS0,115200 earlyprintk rootdelay=1
root=/dev/nfs rw nfsroot=$serverip:$mountpath,nfsvers=4,tcp
ip=$ipaddr:$serverip:$gatewayip:$netmask:$hostname:eth0:off

usb  start
ping $serverip

tftp  $kernel_addr_r $serverip:$tftppath/Image
tftp  $fdt_addr_r    $serverip:$tftppath/nanopi-neo-plus2.dtb

booti $kernel_addr_r - $fdt_addr_r
```

Note: la ligne "`setenv bootargs …`" et les 2 lignes qui suivent doivent impérativement être placées sur une même ligne séparé avec une espace et non sur 3)

- Générer ensuite le fichier `"boot.nfs"` à l'aide de la commande suivante :
  `$ mkimage -T script -A arm -C none -d boot.cmd boot.nfs`
- Copier le fichier `"boot.nfs"` sur la partition vfat BOOT de la carte SD
- Arrêter la cible dans l'U-Boot et changer le script de démarrage de la cible :
  `# setenv boot_scripts boot.nfs`
- Si vous souhaitez conserver cette configuration, taper la commande
  `# saveenv`
- Relancer le système / redémarrer la cible

MA-CSEL1 (Construction de systèmes embarqués sous Linux) – Exercices

Environnement Linux Embarqué

Travail

1. Installez l'environnement de développement sur la machine hôte, selon les instructions ci-dessus, et configurez la cible en mode de développement avec NFS.
2. Installez/configurez SSH pour un accès à distance.
3. Créez un script permettant de générer la carte SD.
4. Testez les différentes méthodes et techniques de débogage proposées par l'environnement Linux. Pour cela, générez les exemples fournis dans le répertoire `"~/workspace/csel1/environment"` en suivant les indications des slides. Ces fichiers se trouvent également sous Moodle.
5. Configurez le noyau Linux afin de monter un usrfs sous ext4 depuis la carte SD, Ce dernier sera monté dans le répertoire `"/usr/local"`. Puis démarrez automatiquement (mode production) un petit programme que vous aurez préalablement placé dans le « usrfs ».
6. Répondez aux questions

Questions

1. Comment faut-il procéder pour générer l'U-Boot ?
2. Comment peut-on ajouter et générer un package supplémentaire dans le Buildroot ?
3. Comment doit-on procéder pour modifier la configuration du noyau Linux ?
4. Comment faut-il faire pour générer son propre rootfs ?
5. Comment faudrait-il procéder pour utiliser la carte eMMC en lieu et place de la carte SD ?
6. Dans le support de cours, on trouve différentes configurations de l'environnement de développement. Qu'elle serait la configuration optimale pour le développement uniquement d'applications en espace utilisateur ?