Projet : Linux embarqué

A. Objectifs

- Compiler le noyau Linux à partir des sources pour la plateforme Versatile Express à base de ARM Cortex-A9
- Développement de pilote de périphérique comme module noyau

B. Préparation de l'environnement

1. QEMU

Qemu est un emulateur de processeur qui permet la simulation de tout un système matériel (machine virtuelle). Qemu supporte plusieurs processeurs cibles (x86, ARM, MIPS,...)

```
sudo apt-get install qemu-system
```

2. GNU toolchain

La chaine de compilation GNU adapté pour l'architecture cible peut être installé via apt-get :

```
sudo apt-get install gcc-arm-linux-gnueabi
```

C. Compilation croisée du noyau Linux

1. Télécharger les sources du noyau linux :

```
wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.2.tar.bz2
tar xjf linux-3.2.tar.bz2
```

2. Initialiser les variables d'environnement ARCH et CROSS_COMPILE

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabi-
```

3. Compiler le noyau

```
cd linux-3.2
make vexpress_defconfig
make all
```

Ceci produira une image compressé du noyau (zImage) sous arch/arm/boot

4. Lancer linux avec QEMU :

```
qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -append "console=tty1"
```

Que se passe t-il à la fin du démarrage du noyau ?

5. Préparation d'un ramdisk

Pour que le noyau exécute quelque chose à la fin de la phase de démarrage, on va créer une simple application de type « hello world » qui va se lancer en mode USER. Pour cela, on utilisera l'option initrd du noyau qui permet de charger un système de fichier minimale au format initramfs. Ce fichier est chargé dans la mémoire vive au démarrage, et la première application que le noyau essaiera d'exécuter est l'application /init

Un exemple de source d'application est le suivant :

```
#include <stdio.h>

void main() {
  printf("Hello World!\n");
  while(1);
}
```

Le programme doit être cross-compilé en version statique (l'exécutable généré contiendra tout les bibliothèques nécessaires). L'exécutable portera le même nom (init) que le programme de démarrage.

```
arm-linux-gnueabi-gcc -static init.c -o init
```

l'image initramfs contenant votre exécutable init peut être crée par la commande cpio :

```
echo init|cpio -o --format=newc > initramfs
```
Le contenu du initramfs peut être vérifié par la commande :
```
cpio -t < initramfs
```
Relancer maintenant le noyau avec l'image initramfs :
```
qemu-system-arm -M vexpress-a9 -kernel linux-3.2/arch/arm/boot/zImage -initrd initramfs -append "console=tty1"
```

6. Utilisation de NFS

Pour faciliter le développement, on va utiliser ce système de fichier via NFS. Ainsi, toute modification du système de fichier sera faite sur le Host (plus facile) et ne nécessite pas la ré-génération de l'image initrd
Procédure
Créer un dossier nfs dans votre dossier de travail
Décompresser l'image rootfs avec sudo
```
sudo tar -xvf rootfs.tar -C nfs
```
Installer NFS server
```
sudo apt-get install nfs-kernel-server
```
Configurer le fichier /etc/exports en ajoutant :
```
/dossier-de-travail/nfs
127.0.0.1/255.255.0.0(rw,no_root_squash,no_subtree_check,insecure)
```
Redémarrer le serveur NFS
```
sudo /etc/init.d/nfs-kernel-server restart
```

Le noyau linux doit être compilé avec les paramètres suivant (activés par défaut)
```
CONFIG_NFS_FS=y (NFS support)
CONFIG_IP_PNP=y (configure IP at boot time)
CONFIG_ROOT_NFS=y (support for NFS as rootfs)
```
et lancé avec les paramètre suivant :
```
root=/dev/nfs
ip=[guest address]
nfsroot=[host address]:[dossier nfs]
```
Avec Qemu, l'adresse par défaut attribuée à la machine simulé est 10.0.2.15. Dans ce cas, la machine simulée voit la machine hôte à l'adresse 10.0.0.2. La commande utilisée est donc :
```
qemu-system-arm -M vexpress-a9 -kernel [chemin-a-image]/zImage -append "root=/dev/nfs nfsroot=10.0.2.2:[chemin-absolue-a-nfs] rw ip=10.0.2.15::10.0.2.1:255.255.255.0 console=tty1"
```

D. Utilisation des modules noyau

Un modules noyau est un moyen d'étendre dynamiquement les fonctionnalités du noyau Linux. L'ajout et la suppression de module est faite en cours d'exécution par l'utilisateur privilégié (root)
- Créer un fichier test.c dans un nouveau dossier :

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int test_init(void){
    printk(KERN_ALERT "Hello, world\n");
    return 0;
}
static void test_exit(void){
    printk(KERN_ALERT "Goodbye\n");
}
module_init(test_init);
module_exit(test_exit);
```

- Créer un fichier Makefile

```
obj-m:= test.o
```

- Compiler le module de test en utilisant le même environnement qui a servi pour compiler le noyau Linux

```
make -C [dossier/source/de/linux] ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
```

```
M=`pwd` modules
```

- Le fichier généré hello.ko doit être copier dans le système de fichier (nfs) de la cible, sous /root/ par exemple
- Tester le module ainsi généré :
  - Pour charger le module : **`insmod test.ko`**
  - Pour afficher la liste des modules chargés : **`lsmod`**
  - Pour décharger le module : **`rmmod test`**

E. Pilote de périphérique

Le pilote de périphérique sera développé comme module kernel. Un pilote de périphérique permet d'abstraire l'accès au matériel. Il fournit donc à l'application une interface de haut niveau. Dans Linux, cette interface passe par le système de fichier. Un périphérique est vue, par l'application, comme un fichier sur lequel un certain nombre d'opérations sont possibles (ouverture, fermeture, lecture, écriture, configuration, etc) . Les fichier spéciaux associés aux périphériques se trouvent dans /dev et peuvent être crées manuellement par la commande :
mknod /dev/<device> [c|b] major minor

1. Pilote de périphérique Test

Le pilote de test qu'on se propose de développer n'interagit pas avec un matériel externe. Il permet simplement de lire/écrire dans un buffer mémoire. (fichier memory.c)
- Compiler et charger le pilote de périphérique
- Créer le fichier spécial associé au pilote (ayant le même major) : **`mknod /dev/memory c 60 0`**
- Charger le pilote : **`insmod memory.ko`**
- Pour tester l'écriture : **`echo –n abcdef >/dev/memory`**
- Pour tester la lecture : **`cat /dev/memory`**

2. Pilote de périphérique à développer

Le pilote de périphérique à développer est un pilote caractère qui permet d'afficher à l'écran une animation (barre de défilement à la Ubuntu).