Cours "Informatique Embarquée"

François Armand

M2 (IMPAIRS, LP, MIC, EIDD...)

Exercice N° 3, 20 Octobre 2017

A rendre avant le Jeudi 2/11/2017 minuit

- 2 séances de TP -

armand@informatique.univ-paris-diderot.fr

Linux, QEMU et BusyBox

1 Consignes:

La remise se fera en suivant les consignes habituelles.

2 Introduction

Le but de ce TP est de créer un petit système basé sur un noyau Linux et utilisant BusyBox. La machine sur laquelle ce système tournera est une machine « fictive » qui sera fournie, émulée par le logiciel **`qemu`**. (http://www.qemu.org).

Pour l'essentiel, il faut donc:

- récupérer les sources d'un noyau Linux, les configurer et compiler le système,
- « Créer un initram disque », en fait un fichier, qui pourra être utilisé comme un ram disque par notre système cible,
- On utilisera initialement un programme de type « hello world » comme programme `init` pour vérifier que tout fonctionne correctement.
- Récupérer les sources de BusyBox, les configurer et les installer sur le disque précédemment créé,
- « Jouer avec les commandes de BusyBox ».

Le TP comporte deux parties : une première partie où le système émulé est une machine Intel 32 bits, et une deuxième partie où qemu fournira une machine basée sur processeur ARM.

Références:

- http://pficheux.free.fr/articles/lmf/hs24/busybox/bb_nutshell.pdf
- http://mgalgs.github.io/2015/05/16/how-to-build-a-custom-linux-kernel-for-qemu-2015-edition.html

Ce TP a été testé sur la machine lucien de l'UFR d'informatique. Il doit être possible de réaliser le TP sur toute machine Linux, notamment sur la machine virtuelle mise à disposition (voir le document sur le site du cours). Une partie de ce TP nécessite l'installation d'une chaîne de compilation croisée d'origine emdebian, il est donc utile pour cette partie d'utiliser la machine virtuelle debian. Je n'ai pas vérifié sur une machine Mac. A bon entendeur, salut !

3 Générer un noyau Linux

On va installer les sources du noyau Linux dans un répertoire. Veillez à ce que **le chemin d'accès à ce répertoire ne contienne pas de caractères espaces**. La construction de Linux

ne supporte pas ce genre de « fantaisies » !

3.1 Parcours balisé

On utilisera la version 4.7.7 dernière testée pour ce TP, mais vous pouvez essayer avec une version plus récente du noyau Linux.

mkdir TP3; cd TP3

wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.7.7.tar.xz

ou

cp /home/armand/Downloads/linux-4.7.7.tar.xz .

Cette commande suppose que vous soyez sur les machines de l'UFR. Si vous êtes sur une autre machine, il vous faudra faire un « `scp` » ou un wget depuis le site http:://www.kernel.org

3.2 Génération du système

3.2.1 Extraire les sources du noyau

Pour générer un noyau Linux, il vous faut maintenant extraire les sources Linux de l'archive (commande tar), si nécessaire, se référer au manuel de la commande tar.

3.2.2 Configurer le système

```
# cd linux-4.7.7
# make help            # vous donnera des indications sur ce que peut faire le make
```

On va générer le noyau Linux dans un répertoire autre que le répertoire qui contient les sources.

```
# mkdir ../linux-477-x86    # ou tout autre nom à votre convenance.
```

Avant de générer le noyau Linux, il faut choisir / modifier / définir sa configuration. Il est conseillé de **ne pas utiliser la configuration par défaut** afin de minimiser les temps de génération du système Linux. Pour gagner un temps précieux on va générer une version (vraiment) très minimale du noyau Linux en utilisant la configuration suivante :

```
# make O=../linux-477-x86 allnoconfig
```

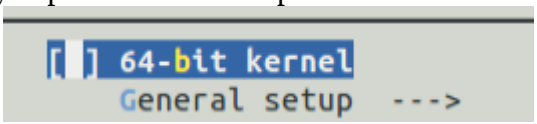
Dans cette version toutes les options sont inhibées. Cependant, cela ne donne pas un système très loquace, et il est donc difficile de savoir / comprendre / analyser ce qui se passe :-(

Il vous faut donc une fois cette commande exécutée, faire un « make menuconfig » pour modifier cette configuration.

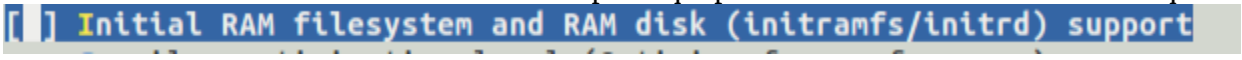
```
# make O=../linux-477-x86 menuconfig
```

Il vous faut alors procéder aux configurations suivantes :

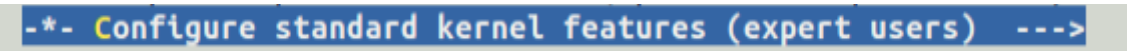
- Ne pas générer un noyau pour 64 bits mais pour 32 bits en décochant la case 64-bit kernel



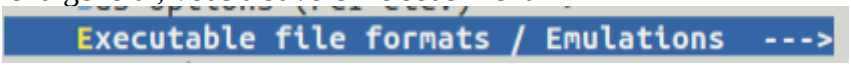
- Dans la section « General Setup » vous définirez le suffixe qui apparaîtra quand on affichera la version du système. Mettez vos initiales, par exemple.
- Dans la même section vous activerez l'option qui permet d'utiliser un « initram » disque



- Dans une la sous-section suivante du « General Setup » vous activerez le support pour « printk »



- Vous voilà promus experts, félicitations !
- Dans le menu général, vous trouverez le sous-menu



pour y activer le support par le noyau Linux des binaires exécutables ELF et des shell scripts.

- Comme vous êtes maintenant devenus experts, il n'y a plus de copié-collé de ce qu'il faut chercher/faire.
- Ensuite, dans un sous-menu (Device Driver, Character Devices, Serial drivers) vous activerez le support de 8250/16550 et vous n'oublierez pas d'activer la console sur ce type de ligne… Sinon, votre système sera muet au démarrage.
- Enfin vous activerez le support des pseudo-file systems proc et sysfs

C'est tout ! Sauvegardez votre configuration et sortez de menuconfig.

Éléments de compte-rendu:

- Indiquez comment vous avez procéder pour effectuer les configurations ci-dessus Donnez le « chemin » de sélection dans le menu de configuration.
- Où se trouve la documentation du noyau Linux dans l'arborescence que vous avez installée ?

3.2.3 Compilez

```
# make -j 4 O=../linux-477-x86
```

Sur la machine lucien peu chargée, la compilation a duré quelques petites minutes (moins de 4). Il est possible qu'en salle de TP, ce soit un peu plus long.

Profitez en :

- Pour chercher quelles sont les différences majeures entre la version 4.7.7 de Linux et la version précédente....

Éléments de compte-rendu:

- Combien de temps a pris votre compilation (donnez des précisions sur l'environnement de génération)?
- Où se trouve le fichier généré, quelle taille fait-il? Le noyau Linux est généré sous différentes formes. Trouvez les et décrivez (très brièvement) leurs formats respectifs.
- Si on avait généré le noyau Linux pour notre machine de développement, que faudrait-il faire ensuite pour « installer » ce noyau et tenter de redémarrer notre machine avec notre nouveau noyau? (Donnez les quelques commandes nécessaires).
- Quelles différences avez-vous trouvé entre la 4.7.7 et la version précédente ? Quelles sources d'information avez-vous utilisées ?

4 QEMU

Qemu est un émulateur de machines. QEMU peut, par exemple, émuler

- un PCx86 sur un PCx86, (32bits et 64 bits)
- un PCx86 sur une machine à base de PPC,
- une machine basée sur un PPC sur une machine x86

- et beaucoup plus encore.

Il est recommandé de comprendre un minimum ce que qemu peut vous permettre de faire :

```
# type qemu
# ls ....../qemu*
# man qemu-system
# qemu-system -help
```

QEMU ne se charge pas seulement d'émuler le processeur, mais il fournit aussi une émulation de certains périphériques associés à une machine: mémoire physique, disques, lignes séries, interfaces réseau, écran graphique, périphériques audio,... La configuration utilisable varie suivant les processeurs émulés. Dans notre cas, nous nous en servirons pour avoir une deuxième machine x86 à notre disposition.

4.1 Oui, mais nous on a généré un noyau Linux pour x86 et notre machine est une 64 bits et on ne peut pas changer le noyau booté dessus...

On peut dans un premier temps, vérifier si notre noyau Linux généré précédemment semble fonctionner raisonnablement:

```
# qemu-system-i386 -nographic -append "console=ttyS0" –kernel TP3/linux-477-x86/ … # à vous de trouver la suite !
```

Le système devrait démarrer. Il y a assez peu de chances pour que la commande ci-dessus vous soit réellement utile. On peut cependant analyser ce qu'il se passe.

Pour sortir de Qemu : CTL-a c puis taper quit.

Éléments de compte-rendu:

- Quel message s'affiche à la fin de l'initialisation du système Linux ? Pourquoi ?

5 Bonjour le monde!

On peut avant de jouer avec BusyBox, créez un programme « init » extrêmement original, et vérifiez si le système fonctionne correctement.

- Créez en C un programme qui répète un message périodiquement chaque seconde pendant 10 secondes avant de se terminer.

```
printf (Hello World!\n");
```

- Attention, la configuration préparéevise une machine 32 bits... Il faut donc générer un binaire pour machine 32 bits...

```
# gcc -m32 hello.c -o hello_32
# file hello_32 ; ./hello_32
```

- Compilez en utilisant une édition de liens statique.
- Vérifiez que votre « hello » fonctionne sur votre machine de développement.
- Lancez le sur Qemu :-)

  # qemu-i386 hellos

- On va préparez une petite arborescence de système de fichiers

```
# mkdir -p ~/TP3/root/sbin
# cp hello ~/TP3/root/sbin/init
```

- Vérifiez que votre fichier « init » a les droits d'exécution et que c'est bien un fichier binaire exécutable 32 bits statique.

- Construisez maintenant un initram disque pour Linux

```
# cd ~/TP3/root
# find . -print0| cpio --null -ov --format=newc \
              gzip -9 > ../initramfs.cpio.gz
```

Éléments de compte-rendu:

- Quelle différence y a-t-il entre qemu-i386 et qemu-system-i386 ? Pourquoi faut-il utiliser qemu-i386 pour « hellos » et qemu-system-i386 pour le noyau Linux ?
- Listez l'arborescence obtenue (format -l)
- Quelle commande avez-vous utilisée pour générer votre programme « init »? Pourquoi ?
- Pourquoi faut-il faire une édition de liens statique ?
- Donnez le résultat de la commande file sur votre programme init / hello.
- Quelle est la taille de cette commande init /hello? Par quelle(s) commande(s) avez-vous trouvé cette information?
- Quelles sont les tailles de ses segments de code et de données, sur disque et en mémoire? Par quelle(s) commande(s) avez-vous trouvé ces informations?
- A quelle adresse en mémoire virtuelle le segment de code sera-t-il placé? Et le segment de données? Et la pile? Par quelle(s) commande(s) avez-vous trouvé ces informations ?

5.1 Une fois le « disque » fabriqué

- Relancez Qemu

```
# qemu-system-i386 -nographic -append "console=ttyS0" –kernel
TP3/linux-477-x86/ … -initrd ../TP3/initramfs.cpio.gz
```

- Si « Hello World! » apparaît, c'est tout bon. Sinon, « Il y a quelque chose cloche là dedans, j'y retourne immédiatement! » (Boris Vian).

Éléments de compte-rendu:

- Quelles erreurs éventuelles avez-vous rencontrées? Pourquoi? Comment avez-vous résolu ces problèmes?
- A quoi sert l'argument « -append » de Qemu? Que peut-on passer comme valeurs à cet argument? Où trouver cette information?
- Que se passe-t-il quand votre programme « init » se termine après avoir affiché « Hello World! »? Pourquoi?

5.2 Un peu de dynamisme!

Nous allons maintenant tenter de recommencer cette expérience mais en utilisant une édition de liens dynamique pour notre programme init.

- Procédez à une édition de liens dynamique de votre programme hello/init.
- Il vous faut déterminer les bibliothèques dynamiques / partagées dont il a besoin et les copier à l'endroit approprié dans le répertoire root créé plus haut.
- Les commandes ldd et cp devraient suffire à votre bonheur. (« C'est quand le bonheur? » -Cali-) La bibliothèque ...gate... n'est pas une vraie bibliothèque, inutile de chercher à la recopier.

- Recréez votre disque et relancez QEMU. Persévérez jusqu'à obtenir un fonctionnement correct, ou épuisement :-(

Éléments de compte-rendu:

- Quelle commande avez-vous utilisée pour générer votre programme « init »?
- Donnez le résultat de la commande file sur votre programme init / hello.
- Quelle est la taille sur disque de cette commande init /hello? Par quelle(s) commande(s) avez-vous trouvé cette information?
- Quelles sont les tailles de ses segments de code et de données, sur disque et en mémoire? Par quelle(s) commande(s) avez-vous trouvé ces informations?
- Donnez le résultat de la commande ldd.
- Indiquez quelles bibliothèques vous avez copié dans votre arborescence root.

6 Une petite compilation croisée

Sur une machine virtuelle debian 9.1 :

```
$ sudo apt-get install -y  emdebian-archive-keyring
$ sudo apt-get install -y gcc-arm-linux-gnueabi
$ gcc-arm-linux-gnueabi hello.c -o hello_arm -static
$ file ./hello_arm
$ ./hello_arm                    # on suppose que qemu n'est pas installé
$ sudo apt-get install -y qemu
$ qemu-arm ./hello_arm
$ ./hello_arm
```

Éléments de compte-rendu:

- Qu'affiche la commande file ./hell_arm ?
- Pourquoi `-static` à la compilation pour cette compilation croisée pour ARM ?
- Que se passe-t-il lors de la première tentative d'exécution ?
- Pourquoi la dernière exécution ./hello_arm se comporte-t-elle différemment de la première ?
- Incluez un appel à pause dans votre hello.c, recompilez et invoquez la commande ps (manuellement) quand ./hello_arm est coincé dans le pause. Donnez le résultat dans votre compte-rendu.

7 BusyBox

« Hello world » est un peu limité comme fonctionnalités pour un système, même si ça pourrait faire un système embarqué très utile: un appareil dont le seul but serait d'afficher « Hello World! » sur un écran quand on le met sous tension! Une sorte de « coucou » suisse électronique. On va donc enrichir le système avec BusyBox.

- Récupérez les sources de BusyBox par exemple

```
# mkdir ~/TP3/BB; cd ~/TP3/BB;
# wget https://www.busybox.net/downloads/busybox-1.24.2.tar.bz2
```

ou toute autre version.

- Configurez BusyBox de manière à ce que le résultat soit généré avec une bibliothèque statique.
- Configurez BusyBox de la manière suivante :
  - `# make menuconfig`
  - Pensez à générer pour un processeur 32 bits (options de build, flag du compilateur positionnés à -m32)
  -
- Compilez
  ```
  # make  # On peut rediriger le résultat comme pour Linux avec O=../bbox-4-x86
  # make install
  ```
- Créez votre nouvelle arborescence racine
  ```
  # cd ~/TP3/root
  # mkdir -p{bin,sbin,etc,proc,sys,usr/{bin,sbin}}
  # cp -av …../_install/* ~/TP3/root
  ```
- Reconstruisez votre initram disque
- Relancez qemu avec ce nouveau disque
- Corrigez les problèmes jusqu'à avoir un shell qui fonctionne.
- Jouez! Attention, vous aurez probablement un clavier qwerty par défaut...

Éléments de compte-rendu:

- Avez-vous rencontré des problèmes? Comment les avez-vous résolus?
- Quelle séquence de commandes avez-vous essayé sur votre machine QEMU?
- Quelle taille fait votre fichier binaire exécutable BusyBox? Quelle est la taille de sa section de code? La taille de sa section de données, sur disque, en mémoire?
- Pourriez-vous configurer busybox de manière à ce que cette commande ait une taille plus réduite que le /bin/bash de votre machine de développement?

7.1 Configuration BusyBox

- Configurez BusyBox de la manière suivante :
  - `# make menuconfig`
  - Pensez à générer pour un processeur 32 bits (options de build, flag du compilateur)
  - Archivage : ne laissez que `tar, gzip, unzip`
  - Éliminez les utilitaires Debian
  - Supprimez `ed` et `patch` des éditeurs fournis par BusyBox
  - Supprimez le support de l'argument `-maxdepth` de la commande `find`
  - Éliminez la commande `mesg`
  - Supprimez le support des commandes « `minix` » de manipulation de systèmes de fichiers.
  - Supprimez la commande `beep` et le support de `splashscreens`.
  - Éliminez le support des différents démons réseau, de IPV6, des utilitaires d'impression et de mail.

7.2 Etendre BusyBox (question totalement optionnelle)

Pour les courageux, puisqu’on en est là… Et si on étendait busybox en lui ajoutant une commande ? Tiens, adaptons colimacon du TP N°2 à busybox :-)