Cours "Informatique Embarquée"

François Armand

M2 SRI/Crypto

Exercice N° 6, 22 Novembre 2012

A rendre avant le 05/12/2012 23H59

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). Qemu sera présenté en cours lors du chapitre consacré aux machines virtuelles, sous réserve de temps.

Pour l'essentiel, il faut donc:

- récupérer les sources d'un noyau Linux, les configurer et compiler le système,
- « Créer un disque », en fait un fichier, qui pourra être utilisé par notre système cible,
- On pourra utiliser 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 ».

La machine émulée par Qemu sera dans ce TP une machine de type PC basée sur un processeur Intel.

Référence: http://pficheux.free.fr/articles/lmf/hs24/busybox/bb_nutshell.pdf

Dans la suite, il vous est possible d'effectuer le TP sur les bases validées (en repartant des archives déjà téléchargées sur les machines de l'UFR), ou de partir en « hors pistes » en travaillant sur des versions différentes de celle testée. Cette possibilité de « hors pistes » est déconseillée aux personnes n'ayant jamais encore généré de noyau Linux. Par contre, suivant le temps dont vous disposez, vous pouvez suivre le parcours balisé, puis recommencer en « hors pistes ».

Il y a donc 2 parcours possibles:

- le parcours balisé (ce qui ne veut pas dire que l'arrivée est garantie)
- le parcours hors pistes qui nécessite des droits super utilisateur sur la machine Linux et n'est donc pas réalisable sur les machines de l'UFR.

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 2.6.26.2 déjà copiée localement:

```
# mkdir TP6; cd TP6
# cp -a ~armand/Downloads/linux-3.12.tar.xz
```

3.2 *Parcours hors pistes*

Les sources du noyau Linux sont disponibles sur le site http:://www.kernel.org.
Téléchargez la version de votre choix dans un répertoire de travail. Par exemple:

```
# mkdir TP6; cd TP6
# wget  https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.12.tar.xz
```

3.3 Génération du système

3.3.1 Extraire les sources du noyau (balisé et hors pistes)

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.

3.3.2 Configurer le système (balisé et hors pistes)

```
# cd linux-3.12
# make help          #  vous donnera des indications sur ce que peut faire le make
# make defconfig     # si vous voulez utiliser la configuration par défaut
# make menuconfig    # si vous voulez configurer manuellement votre système
```

Il est conseillé de ne pas utiliser la configuration par défaut afin de minimiser les temps de génération du système Linux.

3.3.3 Parcours balisé

Il y a un fichier de configuration pour la version 2.6.26.2 à l'emplacement suivant: `~armand/Downloads/config-3-12-fa`, copiez le dans votre répertoire linux dans le fichier « .config». Il faudra quand même exécuter un « make menuconfig» par la suite, cependant il vous est demandé pour vous familiariser avec les mécanismes de configuration du noyau d'effectuer quelques opérations décrites plus loin.

3.3.4 Parcours hors pistes

Si vous avez utilisé une version différente de Linux, il faut alors régler sa configuration soi-même. Pour gagner du temps, éliminez un maximum de fonctionnalités dont nous n'aurons pas besoin : les file systèmes autre que ext2, tout ce qui est réseau, virtualisation, cryptographie... Les drivers inutiles. Éliminez le support des périphériques et matériels inutiles (support du son, ...)

3.3.5 Configurations à faire

A) Faites en sorte que le noyau généré soit « le vôtre »: Modifiez l'option qui permet de définir le suffixe de la version du noyau. Dans le fichier config mis à votre disposition, le

suffixe de la version 3-12 est défini comme étant « -fa ». Retrouvez et modifiez cette option pour personnalisez votre version. On pourra retrouver cette chaîne une fois notre système lancé en invoquant la commande uname -a.

Le mécanisme de configuration du noyau permet:

- d'exclure certaines fonctionnalités,
- d'inclure certaines fonctionnalités dans le noyau dans le noyau
- de les définir comme des modules qui doivent alors être chargés dynamiquement.

B) La configuration définie par le fichier config-3-12-fa, ne définit aucun type de fichiers ext2fs. Modifiez la configuration pour qu'il soit inclus dans le noyau automatiquement. Et pas sous forme de module.

Demandez de l'aide si nécessaire !

Si votre configuration, n'est pas correcte, ce n'est pas très grave. Votre système ne démarrera probablement pas, mais vous devriez obtenir des messages d'erreur qui avec un peu d'aide et d'entraînement vous aideront à obtenir une configuration correcte de votre système.

Éléments de compte-rendu: *(2 pt)*

- Indiquez comment vous avez procéder pour effectuer les configurations ci-dessus (points A, et B). Donnez le « chemin » de sélection dans le menu de configuration.
- Où se trouve la documentation du noyau Linux?

Enfin, au cas où, vous pourrez, en désespoir de cause, obtenir un fichier de configuration fonctionnel en me le demandant.

3.3.6 Compilez

```
# make
```

Sur la machine mono-processeur que j'ai utilisée, le temps d'exécution de cette commande a été de 22 minutes. 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 3.12 de Linux et la version précédente....
- Ou passez du temps sur le TP N°4 (Bugs en C)... *Ah oui, il arrive que l'on ait à travailler sur deux choses en même temps, par exemple, un bug urgent pour dépanner un client et un développement urgent pour la nouvelle version de logiciel qui doit sortir à la fin de la semaine !*

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?
- Quel est le format de ce fichier?
- 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 3.12 et la version précédente ? Quelles sources avez-vous utilisées ?

Au cas où, il est possible d'obtenir une arborescence 3-12-fa précompilée, plus lourde à installer, mais le temps de compilation devrait s'en trouver considérablement réduit.

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
# qemu --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 Si on générait un programme pour processeur arm !

Écrire un programme aussi compliqué que « Hello Qemu! »

Il faut maintenant le compiler pour processeur ARM. Coup de chance, Pierre Letouzey, nous a gentiment installé un compilateur croisé (grâce à emdebian.org). Le répertoire `/usr/bin` doit normalement être connu dans votre variable shell $PATH.

On peut donc compiler :

```
# arm-linux-gnueabi-gcc -static hello.c -o hello_arm
```

Puis vérifier la tête de notre programme !

```
# file hello_arm
```

Reste à exécuter ce programme... Il suffit de trouver une machine avec un processeur ARM:-). Votre téléphone est probablement la machine qu'il vous faut... Plus simple, utilisons Qemu :

```
# qemu-arm hello_arm
```

Waouh ! Non ? Ben moi, si !

Éléments de compte-rendu:

- Pourquoi `-static` à la compilation ?

4.2 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-x86 -nographic -append "console=ttyS0" -kernel TP6/linux.3-12/ … à vous de trouver la suite !
```

Si vous avez généré un noyau pour machine 64 bits :

```
# qemu-system-x86_64 -nographic -append "console=ttyS0" -kernel TP6/linux.3-12/ …  trouvez la suite !
```

Le système devrait démarrer, mais comme il n'a pas de système de fichiers à disposition il ne pourra pas terminer son initialisation de manière utile. Il y a assez peu de chances pour que la commande ci-dessus vous soit réellement utile. Pour terminer qemu, trouvez son pid et tuez le depuis un autre terminal.

Hors-Pistes : Utilisez votre version de système.

Hors-Pistes 2 : Récupérez une version récente de QEMU, générez là et utilisez cette version!

4.3 Fabriquer un disque QEMU

Pour que notre système puisse fonctionner, il faut lui fournir « un disque » (dans notre cas une émulation de disque) contenant un « root file system ». On pourrait aussi utiliser un « ramdisk » .

4.3.1 Parcours balisé

Installez l'archive `~armand/Downloads/genimage.tgz`

```
# mkdir -p ~/TP6/mondisque
# cd ~/TP6/mondisque
# tar -zxvf ~armand/Downloads/genimage.tgz
```

Créez un répertoire de nom `root`: dans le répertoire `mondisque` que vous venez de créer. Ce répertoire `root` est la racine de l'arborescence qui servira de système de fichiers racine à votre noyau Linux :

```
# mkdir -p ~/TP6/mondisque/root
```

Vous êtes maintenant prêts à créer un disque avec un système de fichiers ext2fs qui pourra être utilisé par votre noyau Linux au sein de Qemu. L'arborescence de fichiers sur ce disque sera une copie de ce l'arborescence que vous créerez sous « `root` ». Reste donc à y mettre des fichiers.

4.3.2 Parcours Hors Pistes

Il y a d'autres moyens de parvenir à créer un disque pour Linux sous Qemu. Cela suppose cependant d'avoir les droits de super utilisateur sur la machine de développement, ce qui n'est pas possible sur les machines de la salle de TP. Si vous avez une machine Linux personnelle vous vous pouvez tenter cette séquence d'opérations décrite ci-dessous :

4.3.2.1 *Créer le disque*

Créer un disque pour une machine Qemu :

```
# qemu-img create -f raw mondisque.img 20M

# losetup -f --show mondisque.img  #crée une entrée dans /dev
# fdisk /dev/loop1     # on reprend le nom obtenu par losetup
```

- *fdisk permet de formater un disque*
- *Utilisez le menu pour créer une partition occupant tout l'espace du disque*
- *n, p, 1, RC, RC (où RC signifie « retour chariot/ entrée / enter..)*
- *N'oubliez pas de sauvegarder la déclaration de la partition (w) avant de quitte (q)*
- *On peut relancer fdisk pour vérifier que la partition a bien été créée (fdisk -l)*
- *Notez la taille de la partition ainsi créée.*

Éléments de compte-rendu:

- *Dans la séquence ci-dessus, qu'est-ce qui nécessite d'avoir les droits super-utilisateur?*

4.3.2.2 *Peupler le disque créé*

En fait, ce qui nous intéresse n'est pas le disque « physique » émulé dans le fichier mondisque.img, mais la partition, donc un sous ensemble du disque « physique ». c'est cette partition qu'il va nous falloir initialiser pour pouvoir la peupler avec BusyBox ultérieurement.

```
# losetup -d /dev/loop1  #défaire l'association entre loop1 et mondisque.img
# losetup -f --show -o 32256 mondisque.img
        #crée une entrée dans /dev pour accéder à la partition (et non plus au disque)
# mke2fs -j /dev/loop1 xxxxx
         # xxxx est la taille de la partition obtenue ci-dessus
# mkdir ~TP5/mnt
# mount /dev/loop1 ~TP5/mnt
```

Vous avez un système de fichiers vide dans votre arborescence! Il vous reste à le peupler!

Éléments de compte-rendu:

- *Dans la séquence de commandes ci-dessus, il manque des commandes pour produire un équivalent de ce que fait genimage. Lesquelles?*
- *Dans la séquence ci-dessus, qu'est-ce qui nécessite d'avoir les droits super-utilisateur? (Bis repetita)*

5 Bonjour le monde!

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

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

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

- Attention, la configuration préparée (config-3-12-fa) vise 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.
  - Comment, Pourquoi?
- Copiez votre programme dans root/sbin/init

```
# mkdir -p ~/TP6/mondisque/root/sbin
# cp hello ~/TP6/mondisque/root/sbin/init
```

- Créez un répertoire /dev. Les « devices » sont créés à partir d'un fichier dev.txt

```
# mkdir -p ~/TP6/mondisque/root/dev
```

- Puis lancez la commande genimage pour obtenir un disque par exemple, baptisé `test.img`

```
# cd ~/TP6/mondisque
# ./genimage
```

Vous trouverez dans ce répertoire un fichier dev.txt utile pour définir d'autres device pour votre système si nécessaire.

Vous avez maintenant un disque prêt à l'emploi. Félicitations !

Éléments de compte-rendu:

- Listez l'arborescence obtenue (format -l)
- Quelle commande avez-vous utilisée pour générer votre programme « init »? Pourquoi ?
- 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?
- 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 ?
- Pourquoi faut-il utiliser une édition de liens statique?
- Que faudrait-il faire pour utiliser une édition de liens dynamique?

5.1 Une fois le « disque » fabriqué

- Relancez Qemu

```
# qemu-system-x86 -nographic -kernel
TPT/linux.3.12/arch/x86/boot/bzImage -hda tst.img -append
'root=/dev/hda1 console=ttyS0'
```

- 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 valeur 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 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.

6.1 Première étape

- Récupérez les sources de BusyBox par exemple
  `# mkdir ~/TP6/BB; cd ~/TP6/BB;`

  `# wget http://www.busybox.net/downloads/busybox1.16.0.tar.bz2`
  #ou une autre version.
- Configurez BusyBox de manière à ce que le résultat soit généré avec une bibliothèque dynamique.
- 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.
- Compilez
  `# make`
- Pensez aux bibliothèques dynamiques!
- Installez votre BusyBox dans votre système de fichiers qui sert à fabriquer votre disque Linux

```
# make CONFIG_PREFIX=~TP6/mondisque/root install
# cd ~TP6/mondisque; ./genimage
```

- Relancez votre machine sous QEMU.
- Jouez! Attention, clavier qwerty par défaut...

Éléments de compte-rendu: *(2 pt)*

- 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?

6.2 Un peu plus compliqué

- Créez le(s) répertoire(s) suivant dans votre arborescence root:
  - root/etc/init.d
  - root/proc
- Créez le fichier:
  - root/etc/init.d/rcS
  - avec le contenu suivant:

```
#!/bin/sh
loadkmap < /etc/french.kmap
mount -t proc /proc
mount -o remount,rw /
mount -a
```

- Sur un terminal de votre machine de développement exécutez la commande suivante:

```
...../genimage/root/bin/dumpkmap >
.../genimage/root/etc/french.kmap
```

- Rendez le script rcS exécutable

```
# chmod a+rx
```

- Reconstruisez votre disque et relancez votre système.