TP 1 : miniboot

Systèmes d'exploitation L2 MI / DLMI 2025-2026 *Léo Henry*

Le but de ce TP est de **booter** (démarrer) un mini-système Linux, puis de réviser les commandes shell sur ce système. Prendre des notes et déposer un compte-rendu de ce TP sur Ametice pour le 30 janvier. Ce compte-rendu n'a pas besoin d'être long, il reprendra et expliquera le résultat des commandes principales effectuées.

1) Le Noyau

Nous allons créer un mini-système d'exploitation sur lequel booter (par virtualisation). On commence par créer un noyau Linux.

Créer un répertoire de travail « miniboot »

```
$ mkdir miniboot # crée le dossier
$ cd miniboot    # déplace le shell dans le dossier
```

Téléchargez une version LTS (Long Term Support) récente du code source de Linux (cela peut prendre un peu de temps…, vous pouvez **lire la suite de l'énoncé en attendant**)

```
$ git clone --depth=1 --branch=v6.12 git://git.kernel.org/pub/scm/linux/
   kernel/git/torvalds/linux.git
```

On ne va pas voir ici les détails de la compilation du noyau, nous allons seulement utiliser quelques raccourcis afin de créer un **noyau minimal**.

```
$ cd linux
$ make tinyconfig
$ make menuconfig
```

La dernière commande vous place dans l'environnement où l'on peut configurer toutes les options de compilation, pas de panique : lire ce qui est écrit en haut !

Avec les flèches, la touche ESPACE pour sélectionner et TAB pour le menu du bas, vous allez activer les deux suivantes :

- Activer `64-bit kernel` avec la barre ESPACE
- `General setup > Initial RAM filesystem and RAM disk (initramfs/initrd) support` activer
- `General setup > Configure standard kernel features (expert users)`, activer `Enable support for printk`
- `Device Drivers > Character devices`, activer `Enable TTY`
- `Executable file formats > Kernel support for ELF binaries`, activer
- `Executable file formats > Kernel support for scripts starting with #! in the kernel`, activer
- Sauvegarder avec `Save`

Les indications de compilation nécessaire sont désormais bien dans le fichier .config, on peut donc lancer la compilation (là aussi, cela va prendre un peu de temps…)

```
$ make -j8
```

Si vous obtenez le message Kernel : arch/x86_64/boot/bzImage is ready (#1), bravo vous avez un “noyau” (dans un format compressé) dans ce sous-répertoire, sur lequel nous allons pouvoir booter.

Si cela ne compile pas bien, il est possible d'utiliser ce noyau.

Pour se faire, copiez le fichier (par exemple avec wget) dans `arch/x86/boot` puis créez le dossier `arch/x86_64/boot` et créez-y un lien symbolique vers le bzImage (par exemple avec ln -s).

2) Le Processus `init`

On lance notre noyau avec l'émulateur qemu.

```
$ qemu-system-x86_64 -net nic -nic none -kernel arch/x86_64/boot/bzImage
```

On obtient, à la fin, un message Kernel panic, no working init found. En effet, si on a effectivement bien un noyau, cela ne suffit pas pour avoir un système : il nous faut un premier programme, un **premier processus**. Le nom général de celui-ci est init.

Pour cette première séance, nous n'allons pas créer nous-même ce programme, mais utiliser un système minimal conçu pour cela : busybox.

```
$ cd ..
$ git clone --depth=1 --branch=1_37_0 git://git.busybox.net/busybox
$ cd busybox
```

Il faudra ici aussi configurer le système de compilation. Prendre la configuration par défaut

```
$ make defconfig
```

Il faut cependant ajouter une configuration pour obtenir un binaire statique (pourquoi ?). Cela se fait dans le propre menuconfig de busybox.

```
$ make menuconfig
```

Activer l' option `Settings > Build static binary` et désactiver l'option `Networking Utilities > tc (8.3 kb)` puis lancer la compilation avec un `make`.

3) Le Système de fichiers de démarrage

3.1) Créer `initrd`

Après avoir créé init, nous allons devoir mettre en place son environnement, c.à.d. un système de fichier particulier, qui n'existera qu'en RAM. Cela s'appelle un « initial ram filesystem » (en abrégé initramfs).

```
$ cd ..
$ mkdir initramfs
```

Tout ce qui se trouve dans ce répertoire sera fourni au noyau pour lancer init. Nous allons donc créer un répertoire bin et y copier le **fichier** busybox se trouvant dans le dossier du même nom.

Une fois que cela est réalisé, nous allons indiquer à busybox de s'exécuter comme s'il était un shell bin/sh :

```
$ cd initramfs/bin
$ ln -s busybox sh
$ cd ..
```

Regardez la page du manuel de `ln` (avec par exemple `man ln`) Que fait la commande `ln` ? Son option `-s` ?
Nous pouvons préparer notre image disque avec la commande suivante :

```
$ find . -print0 | cpio --null --create --verbose --format=newc | gzip --
    best > initrd
```

Notez la présence de **piping** ($A \mid B$ signifie que B prend en argument le retour de A) et la commande `>` qui spécifie un fichier dans lequel écrire le résultat.

En vous servant du manuel pour les différentes commandes, pouvez-vous expliquer ce que fait la commande ci-dessus ?

3.2) Relancer le noyau

Maintenant qu'initrd est créé, on lance le noyau avec :

```
$ cd ..
$ qemu-system-x86_64 -nec nic -nic none -kernel linux/arch/x86_64/boot/
    bzImage -initrd initramfs/initrd
```

Si qemu capture notre souris, faire Ctrl+Alt+G. Il faut noter également que le clavier est en QWERTY...

On peut faire un test avec echo Hello. Cependant, il semble manquer un environnement complet pour busybox (par exemple, testez ls). On va compléter notre image initrd.

Créer un fichier init dans le réservoir bin et le remplir avec :

```
#!/bin/busybox sh

/bin/busybox --install -s

exec /bin/sh
```

Reconstruire l'image :

```
$ find . -print0 | cpio --null --create --verbose --format=newc | gzip --
    best > initrd
```

et relancer qemu en ignorant les messages No such file or directory pour l'instant.

4) Le shell à la busybox

4.1) Révision de commandes Shell

Si besoin, reprenez vos TDs/TPs précédents sur les commandes Shell (ou cherchez les commandes les plus utilisées et visitez le manuel).

4.2) Premier contact

Testez le shell avec une commande simple (typiquement busybox ls). Si vous essayez de vous déplacer un peu, vous vous rendrez compte que le clavier est en QWERTY.

On peut modifier la configuration du clavier avec la commande

```
# busybox loadkmap < /etc/fr.map
```

mais on n'a pas encore le fichier fr.map. Créez le répertoire etc dans initramfs et copiez le fichier fr.map. Rereconstruisez ensuite initrd et relancez le système avec ce nouveau initrd.

4.3) Test du shell

Quel est la particularité de l'environnement busybox dans lequel vous êtes ? Quelles commandes ne sont pas présentes dans busybox ? Lesquelles ne fonctionnent pas bien.

Notez que certains programmes ne sont pas accesibles directement (ie ni intégrées au shell, ni installés dans /bin) mais accessible via

```
# busybox command
```

Par exemple

```
# busybox whoami
```

Vous pouvez voir l'ensemble des commandes accessibles via busybox --help. Il y a, par exemple, un mini éditeur de fichier, comment s'appelle-t-il ?

4.4) Compléter l'installation

Pour finir, compléter votre script init puis (rerere)reconstruire votre initrd afin de rendre une partie de ces opérations automatiques :

- mise en place du clavier fr
- configurations pratiques diverses
- ...

Si l'on voulait ajouter d'autres programmes que ceux fournis par busybox, comment faudrait-il faire ?