Alain DIAS
Benoit Doll

Rapport TP5

Temps estimé : 10 heures
Temps de réalisation : 11 heures 30

Pour la réalisation de ce TP, nous avons choisi de suivre le parcours balisé sur les machines de l'UFR.

1 Générer le noyau linux

Après avoir téléchargé et extrait le noyau linux version 3.12, nous avons procédé aux configurations demandées. De plus, une documentation du noyau nous est fourni dans le dossier du noyau. Cette documentation est donc disponible dans le dossier *Documentation*.

1.1 Modification du suffixe

Pour effectuer la modification du suffixe de la version du noyau, nous avons fait un *make menuconfig* après avoir copié *config-3-12-fa* et l'avoir renommé en *.config*. Une fois à l'intérieur de la configuration, à l'intérieur du menu *general setup* il nous a été possible de modifier *local version* afin d'y inscrire un nom personnalisé.

1.2 Inclure ext2fs

Toujours dans le menu de configuration, un menu *File systems* est disponible. Dans ce menu nous avons validé l'inclusion de *Second extended fs support*. Pour l'inclure dans le noyau, il fallait choisir entre appuyer sur Y ou sur M. Cependant M aurait eu comme effet de l'inclure sous forme de module... Nous avons donc inclus avec Y.

1.3 Compilation

Sur une machine de l'UFR, la compilation du noyau avec make a duré environ 15 minutes. A la fin de la compilation, nous avons eu le resultat suivant :

Setup is 16108 bytes (padded to 16384 bytes). System is 1461 kB CRC dcae56ba Kernel : arch/x86/boot/bzImage is ready (#1)

Le fichier généré se trouve donc dans le dossier arch/x86/boot/, se nomme bzImage et fait 1477 kilo-octets. Avec la commande file sur ce fichier on a :

linux-3.12/arch/x86/boot/bzImage : Linux kernel x86 boot executable bzImage, version 3.12.0-alain-dias (dias@tangle-up) #1 Fri Nov 28 11 :36 :00 CET 2, RO-rootFS, swap_dev 0x1, Normal VGA

Ce fichier a donc le format bzImage.

Si l'on avait généré le noyau Linux pour notre machine de développement, il nous aurait fallu exécuter un *make install* qui aura pour effet de copier les fichiers nécessaire dans le dossier */boot/* de notre machine. Ensuite nous pouvons redémarrer la machine avec la commande *reboot*. Pour finalement vérifier que le noyau utilisé est bien celui que nous avons installé, une utilisation de la commande *uname -a* nous sera utile.

1.4 Différence avec la version précédente

Voici une liste non exhaustive des différences entre la version 3.12 du kernel de linux et la version précédente :

- Ext 4 (amélioration de performance)

- F2FS (amélioration de performance)

- Z-RAM

- la gestion de la mémoire (l'API de DRM garantie l'isolation entre les différents clients)

Voici la l'adresse où on peut trouver la liste complète :

- http ://linuxfr.org/news/sortie-de-linux-3-12

2 Qemu

2.1 Générer un programme pour processeur ARM

Dans un premier temps, nous avons écrit un programme qui affiche "Hello world !". Nous l'avons compilé avec le compilateur croisé. La commande file sur ce fichier nous donne :

hello_arm : ELF 32-bit LSB executable, ARM, version 1 (SYSV), statically linked, for GNU/Linux 2.6.18, BuildID[sha1]=0x35b35dc3e497320bc5a308e95db376dd66f699 not stripped

L'executable est donc destiné à un processeur ARM.

Il fallait utiliser l'option -static afin d'inclure les librairies nécessaire au fonctionnement du programme à l'intérieur de l'executable. Puisque nous devons passer par qemu pour tester notre programme, il faut que qemu dispose des librairies. En effet, la machine émulée par qemu n'a pas de disque et donc ne dispose pas de librairies dans un dossier */lib/* par exemple.

2.2 Bonjour le monde

2.3 Compilation statique de init

Pour compiler le programme hello world en incluant les librairies dans le binaire, nous avons utilisés la commande *gcc -m32 hello.c -static -o init*. Compiler avec l'option *-static* nous permet de ne pas nous soucier d'ajouter les librairies dans le dossier */lib/* du disque que nous avons crée pour qemu.

On peut vérifier cela grâce à la commande file :

init : ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, for GNU/Linux 2.6.26, BuildID[sha1]=0xba032c7b78ffd7394eb355ac2f8c46f218c4cd6a, not stripped

Grâce à *du -h init*, on voit que le programme init fait 594 kilo-octets.

Pour connaître la taille des segments de code et de données sur le disque et en mémoire, on a exécuté la commande *size init*.

text	data	bss	dec	hex	filename
528650	1956	7044	537650	83432	init

On a donc :

-taille de sa section de code : 528650 octets
-taille de sa section de données sur disque : 9000 octets
-taille de sa section de données en mémoire : 1956

Grâce a la commande *readelf* on trouve :

-Avec l'option *-h* : Entry point address : 0x8048140. C'est à cette adresse virtuelle que le segment de code est placé.
-Avec l'option *-l* : On trouve que la pile est à l'adresse virtuelle 0x00000000 et que le segment de données est à l'adresse virtuelle 0x080ca1ec.

2.4 Creation du disque (version statique de init)

Voici l'arborescence du disque executant notre programme au démarrage :

root/
| | dev/
| | sbin/
| | | init

Si l'on avait voulu utiliser une compilation dynamique, il aurait fallu ne pas mettre d'option *-static* puis copier toutes les librairies dont le code dépend dans un dossier */lib/* en veillant à garder la bonne arborescence.

Nous n'avons rencontré aucune erreur lors du lancement de qemu. Il nous a juste fallu adapter la commande pour correspondre à notre cas :

qemu-system-x86_64 -nographic -kernel linux-3.12/arch/x86/boot/bzImage -hda mondisque/test.img -append 'root=/dev/hda1 console=ttyS0'

L'argument -append permet de passer des paramètres au noyau Linux. Dans notre cas nous avons spécifié que le système de fichier à monter comme racine est */dev/hda1* et nous avons redirigé la console sur *ttyS0*. On peut passer comme valeur par exemple init=Fichier. Cette valeur permet de spécifier quel est le programme à lancer après le boot. Cependant, sans cette valeur, c'est le fichier init se trouvant dans sbin qui sera choisi.

Une fois notre programme terminé, la machine reste en marche mais ne fait plus rien. Il faut éteindre à la main (en utilisant kill dans notre cas) pour quitter qemu.

2.5 Compilation dynamique de init

Pour compiler le programme hello world dynamiquement, nous avons utilisés la commande *gcc -m32 hello.c -o init*.

On peut vérifier cela grâce à la commande file :
init : ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.26, BuildID[sha1]=0xd64f88212941f46dcce2b74e
not stripped

L'execution de la commande size sur le fichier init nous donne :

text	data	bss	dec	hex	filename
1126	292	4	1422	58e	init

On a donc :

-taille de sa section de code : 1126 octets
-taille de sa section de données sur disque : 296 octets
-taille de sa section de données en mémoire : 292

Grâce à la commande *du -h init* on voit que le programme fait maintenant 8 kilo-octets.

La commande *ldd init* nous donne :
linux-gate.so.1 => (0xf774f000)
libc.so.6 => /lib/i386-linux-gnu/i686/cmov/libc.so.6 (0xf75bd000)
/lib/ld-linux.so.2 (0xf7750000)

Nous avons donc copié les librairies libc.so.6 et ld-linux.so.2 dans notre arborescence comme suit :

root/
| dev
| lib
| | i386-linux-gnu
| | | cmov
| | | | libc.so.6
| | ld-linux.so.2
| sbin
| | init

3 BusyBox

3.1 Problèmes rencontrés

Nous avons rencontrés durant le processus de configuration de busybox divers problème. Premièrement, il y a eu un bug lors de la configuration du réseau dans le menu. Il était possible d'entrer dans le menu mais impossible d'en sortir. Pour contourner ce problème, il nous a fallu modifier le fichier .config à la main.
Ensuite, nous avons suivi la procedure trouvée sur le forum de didel afin de résoudre un problème au niveau de l'edition de lien avec le make. Il fallait executer *export LDFLAGS='-m32'* avant de lancer le make.
Finalement, une fois le disque crée, il nous a été signalé que *tty2*, *tty3* et *tty4* étaient introuvable. Nous avons donc modifié le fichier *dev.txt* afin d'y ajouter les lignes :

/dev/tty2 c 600 0 0 4 1
/dev/tty3 c 600 0 0 4 1
/dev/tty4 c 600 0 0 4 1

3.2 Commandes essayés

Tout d'abord on a essayé de verifier que les commandes patch, ed et mesg n'existaient pas. On a ensuite essayé de créer un dossier, mais le système de fichier est *read-only*, c'était donc impossible.
La navigation dans le système de fichier quant à elle était fonctionnelle (cd, ls). Nous avons également utilisé la commande *du -h* pour trouver la taille du binaire de busybox. Nous voulions aussi utiliser la commande size, mais elle n'était pas disponible.

3.3 Taille busybox

Le fichier binaire de busybox fait 696 kilo-octets.

Lors de l'execution de la commande size sur le fichier binaire on obtient :

text	data	bss	dec	hex	filename
704873	2049	8984	715906	aec82	busybox

On a donc :

-taille de sa section de code : 704873 octets
-taille de sa section de données sur disque : 11033 octets
-taille de sa section de données en mémoire : 2049

La configuration que nous avons réalisé pour *busybox* nous a déjà produit un binaire plus petit que le *bin/bash* de notre machine de developpement qui fait 960 kilo-octets. C'est donc possible.