NMV - NI436

TP 1 – Programmation de module dans le noyau Linux

Julien Sopena

Octobre 2015

Mise en garde : au cours de ce TP nous allons implémenter et charger des modules. Ces modules ont un accès à la totalité de la mémoire du noyau et peuvent donc corrompre son fonctionnement normal. Les conséquences peuvent être irrémédiables sur le système et sur vos données (corruption du système de fichier). C'est pourquoi il est recommandé de travailler sur un système test indépendant de votre machine de travail.

Exercice 1 : Environnement de développement de l'UE.

Question 1

Dans le cadre de cette UE, nous allons utiliser une instance de la machine virtuelle **qemu** comme plateforme de développement. Pour commencer, **comme avant chaque début de TP**, copiez depuis /usr/data/sopena/nmv, l'image *nmv-archlinux.img* dans `/tmp`. Cette image contient une distribution archlinux avec un noyau *Linux* et un compte root sans mot de passe. Que ce passe-t-il lorsqu'on la lance avec la commande suivante :

```
qemu-system-x86_64 -hda nmv-tp.img
```

Question 2

Votre image étant stockée dans le `/tmp`, elle sera perdue à chaque redémarrage de la machine. Vous allez donc ajouter un autre disque qui sera monté dans `/root` (le répertoire de travail de l'utilisateur *root*). Générez une image personnelle à l'aide des commandes suivantes :

```
dd bs=1M count=50 if=/dev/zero of=~/myHome.img
/sbin/mkfs.ext4 -F ~/myHome.img
```

Question 3

Récupérer maintenant le script qemu-run.sh de lancement qui se trouve dans /usr/data/sopena/nmv. Ouvrez le puis corrigez si besoin les chemins menant au différent fichier en fonction de votre configuration. Vous pouvez maintenant lancer votre vm et vous connecter en tant que root (pas de mot passe).

Exercice 2 : Configuration et compilation du noyau Linux

Nous allons maintenant créer notre propre version du noyau sur la machine hôte et l'utiliser pour lancer notre VM. Cette technique permet non seulement d'éviter de compiler le noyau dans la VM, mais simplifiera aussi le débogage (voir TP 03)).

Pour commencer décompresser l'archive `linux-4.2.3.tar.xz` dans le répertoire `/tmp`

```
tar -xvJf /usr/data/sopena/nmv/linux-4.2.3.tar.xz -C /tmp/
```

Question 1

Pour simplifier ce TP et accélérer la compilation, nous avons préparé une configuration allégée du noyau. Copiez cette configuration dans le répertoire de votre noyau sous le nom **.config**.

```
cp /usr/data/sopena/nmv/linux-config-nmv /tmp/linux-4.2.3/.config
```

Accédez à la configuration du noyau linux à l'aide de la commande **make nconfig**. Familiarisez vous avec l'interface en parcourant quelques options. Dans quelle catégorie allons nous trouver des options de débogage que nous pourrons utiliser dans le TP 03 ?

Question 2

Compiler le noyau est une étape simple mais qui peut s'avérer très longue. Si optimiser sa configuration permet de limiter la taille du code compilé, il est nécessaire d'utiliser au mieux les ressources disponibles.

Commencez par trouver le nombre de cœurs de votre machine. Quelle doit alors être la valeur du paramètre `-j` lorsque vous compilerez votre noyau.

Question 3

Lancez la compilation de votre noyau puis affichez les informations de votre image à l'aide de la commande `file`

```
file arch/x86/boot/bzImage
```

Question 4

Comme pour le script `qemu-run.sh` corrigez si besoin les variables du script qemu-run-externKernel.sh Puis utilisez le pour démarrer une machine virtuelle et affichez le nom du noyau en cours d'exécution à l'aide de la commande **uname -r**. Pouvez vous être sûr qu'il s'agit de votre noyau ?

Question 5

Dans la section *General setup*, modifiez l'option de configuration qui permet de modifier le nom du noyau pour qu'il se termine par **-nmv**.

Recompilez votre noyau, démarrez une machine virtuelle sur celui-ci et vérifiez à l'aide de la commande **uname -r** qu'il s'agit bien du votre.

Question 6

Dans votre machine virtuelle, affichez la liste des modules chargés. En étudiant la configuration du noyau, expliquez le résultat obtenu.

Exercice 3 : Le processus *init*

Le but de cet exercice est de comprendre le rôle du processus init dans le démarrage d'un système Linux, et notament de comprendre qu'il s'agit d'un programme comme un autre qui peut être remplacé par n'importe quel executable.

Question 1

Pour commencer implémentez **dans la VM** un programme `hello` qui attendra 5 secondes pour se terminer après le classique *Hello World*.

Question 2

Le kernel dispose d'une option `init=xxx` pour modifier le binaire *init* à exécuter. Cette option peut être définie dans la configuration du boot loader (ici grub), mais nous allons plus simplement le faire en passant par notre script `qemu-run-externKernel.sh`. Modifier ce dernier pour que le noyau exécute votre programme `hello` comme processus *init*.

Question 3

Expliquez pourquoi le système fini par crasher.

Question 4

Puisque l'on peut remplacer l'*init* original par n'importe quel programme, il est possible de lancer un shell en tant que processus 1. Tester en lançant quelques commandes cette astuce qui permet dans bien des cas de récupérer un système corrompu.

Si la plupart des commandes sont actives, pourquoi ne peut-on pas lancer directement une commande `ps` ?

Question 5

Corrigez le problème et testez les commande `ps aux` et `pstree 0`.

Question 6

Pour finir nous allons tenter de retrouver un fonctionnement normal en finalisant le processus de démarrage. Trouvez un moyen de lancer le script `init` original depuis votre shell.

Exercice 4 : Comprendre le fonctionnement du *initramfs*

Dans cette exercice nous allons étudier le principe de l'*initramfs* et voir comment ce mécanisme de chargement est à la base de nombreuses distributions.

Question 1

Pour commencer nous allons analyser le contenu de l'`initramfs` de votre distribution linux. Après avoir créé un repertoire dans `/tmp`, désarchivez l'`initramfs` présent dans le répertoire `/boot` grâce à la commande suivante. En parcourant l'arborescence obtenu vous remarquerez la présence d'un exécutable `init` à la racine de l'archive.

```
cd /tmp/test zcat /boot/initramfs-linux.img | cpio -i -d -H newc -no-absolute-filenames
```

Question 2

Nous allons maintenant créer notre propre initramfs. Après avoir créé un repertoire `racine` contenant une version de votre programme `helloWorld` nommé `init`, vous allez générer une archive cpio à l'aide des commandes suivantes :

```
cd racine find . | cpio -o -H newc | gzip > ../my_initramfs.cpio.gz
```

Question 3

Pour tester votre archive il vous suffit d'utiliser l'option `-initrd` de qemu.

Exercice 5 : Mes premiers modules

Question 1

Copiez depuis `/usr/data/sopena/nmv/TP-02` les sources du module *HelloWorld* et compiler les. Parmi les fichiers générés quel est celui/ceux qui seront chargés dans le noyau ?

Question 2

A l'aide de la commande **$(info xxx)** essayez de comprendre comment sont exécutées les lignes du *Makefile*.

Question 3

Chargez puis déchargez le module *HelloWorld*, puis listez les endroits où l'on peut lire le message **"Hello world**

Question 4

À partir du module *HelloWorld*, écrivez un module *HelloWorldParam* qui utilisera deux paramètres "whom" (une chaîne de caractères) et "howmany" (un entier) pour afficher :

```
insmod helloWorldParam.ko whom=julien howmany=3
   (0) Hello, julien
   (1) Hello, julien
   (2) Hello, julien
rmmod helloWorldParam
   Goodbye, julien
```

Question 5

Tester l'affichage des informations de module obtenu grâce à la commande `modinfo`. Vous modifierez si besoin votre code pour avoir une description de chaque paramètre.

Question 6

En modifiant si besoin le module *HelloWorldParam*, utilisez l'interface `/sys` pour changer (à posteriori) l'affichage de son déchargement.