

## PNL - MU4IN402

### TP 04 – Mes premiers modules

Redha Gouicem, Maxime Lorrillere et Julien Sopena

février 2020

**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 irréversibles 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.

**Recherche dans les sources :** Pour l'ensemble de ce TP, et les suivants, vous pouvez utiliser comme moteur de recherche le site [elixir.bootlin.com](http://elixir.bootlin.com) qui indexe les sources du noyau. Il repose sur le projet libre [lxr](http://lxr), un utilitaire général d'indexation de code source et de références croisées qui permet la navigation au sein du code source en utilisant une technologie web.

#### Exercice 1 : Mes premiers modules

**Environnement :** ce TP reprend l'environnement de programmation mis en place lors du troisième TP. Il suppose entre autre l'utilisation dans qemu de l'image *pnl-tp.img*, ainsi que l'existence d'un fichier personnel *myHome.img* correspondant au `/root`. Il suppose aussi que vous ayez un fichier de configuration pour le noyau Linux 4.19.3 et que vous ayez dans `/tmp` un exemplaire des sources.

#### Question 1

Copiez depuis `/usr/data/sopena/pnl/TP-04` les sources du module *helloWorld* et compilez-les. Parmi les fichiers générés, quel est celui/ceux qui seront chargés dans le noyau ?

#### Question 2

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

#### Question 3

En vous inspirant 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 4

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 5

En modifiant si besoin le module *helloWorldParam*, utilisez l'interface `/sys` pour changer (a posteriori) l'affichage de son déchargement. Ainsi vous pourrez vous dire "Hello" et dire "Goodbye" à votre binôme.

### Exercice 2 : Modification d'une variable du noyau à l'aide d'un module

#### Question 1

La variable globale `init_uts_ns` est utilisée dans le noyau pour stocker les informations retournées par l'appel système `uname`. Dans les sources du noyau, trouvez où est initialisée cette structure et analysez les champs qu'elle contient. Pour cela, vous pouvez être amenés à utiliser le site d'indexation du noyau conseillé dans le préambule de ce sujet.

Est-ce que cette variable peut être accédée par tous les modules ?

#### Question 2

Créez un module `uname` dont le rôle sera de modifier le nom du noyau lors de son chargement.

#### Question 3

Lors de son déchargement, il est impératif que le module restaure le nom d'origine. Pourquoi ?

### Exercice 3 : Les limites des modules

Les modules permettent d'étendre les fonctionnalités d'un noyau sans même avoir besoin de redémarrer la machine. Cependant, ils ne peuvent pas tout faire, et il est parfois nécessaire de modifier directement le code du noyau et de le recompiler pour implémenter la fonctionnalité désirée. Dans cet exercice, les modules que nous vous demandons de créer **ne peuvent fonctionner sans modification du noyau**.

#### Question 1

On souhaite réaliser un module qui affiche des informations sur les *superblocks* chargés en mémoire par le noyau, par exemple l'UUID du superblock et son type de système de fichiers. Pour cela, il est nécessaire de parcourir la liste des `struct super_block`. Pour faciliter cette opération, le noyau propose la fonction

`iterate_supers`, qui effectue ce parcours à votre place et exécute la fonction passée en paramètre pour chaque superblock.

À l'aide de cette fonction, créez le module `show_sb` qui parcourt la liste des superblocks et produit un affichage similaire à celui-ci :

```
uuid=00000000-0000-0000-0000-000000000000 type=rootfs
uuid=00000000-0000-0000-0000-000000000000 type=bdev
...
uuid=f842acdd-39c7-4678-bb9f-670592fef78c type=ext4
uuid=00000000-0000-0000-0000-000000000000 type=tmpfs
```

Produire un tel affichage n'est pas aisé, mais la fonction `printk` permet formater les données suivant un grand nombre de formats. Vous trouverez tous les spécificateurs associés dans la documentation du noyau [How to get printk format specifiers right](#).

Vous trouverez les informations sur la structure `struct super_block` et sur la fonction `iterate_supers` dans le fichier `include/linux/fs.h`. Comme pour les exercices précédents, vous pourrez vous aider de LXR pour trouver des exemples d'utilisation.

## Question 2

On souhaite désormais créer le module `update_sb`. Basé sur le code du précédent module, celui-ci doit afficher, lors de son chargement, la liste des superblocks d'un système de fichiers particulier passé en paramètre (ext4 par exemple). Il doit également être capable d'afficher la date de son dernier affichage.

Il va donc être nécessaire de conserver individuellement cette date pour chaque superblock. En effet, une partition `ext4` peut avoir été ajoutée après l'affichage des infos des superblocks de ce type. Dans ce cas, sa date diffère de la date des autres superblocks `ext4`.

Pour cela, vous pourrez utiliser la fonction `get_fs_type` pour obtenir une structure qui représente un type de système de fichiers, et la fonction `iterate_super_type` qui parcourt la liste des superblocks d'un type particulier.

**Important :** l'objet retourné par `get_fs_type` a un compteur de références. Vous **devez** rendre cette référence quand vous avez terminé, en utilisant la fonction `put_filesystem`.

Vous pourrez aussi utiliser la fonction `ktime_get` qui permet d'obtenir la date courante.

Au final, vous devriez obtenir un affichage similaire à celui-ci :

```
# insmod ./update_super.ko type=ext4
uuid=f842acdd-39c7-4678-bb9f-670592fef78c type=ext4 time=0.000000000
uuid=3413b1a2-7c1f-429e-9de8-1b4f52f23fd3 type=ext4 time=0.000000000
# rmmod update_super.ko
# insmod ./update_super.ko type=proc
uuid=00000000-0000-0000-0000-000000000000 type=proc time=0.000000000
# rmmod update_super.ko
# insmod ./update_super.ko type=ext4
uuid=f842acdd-39c7-4678-bb9f-670592fef78c type=ext4 time=1445346469.983057934
uuid=3413b1a2-7c1f-429e-9de8-1b4f52f23fd3 type=ext4 time=1445346469.983058280
```

## Exercice bonus 4 (rootkit 1) : Cacher un module

Dans cet exercice, on cherchera à dissimuler la présence d'un module dans la mémoire du noyau. Les techniques employées permettront d'étudier la structuration des objets du noyau, leur référencement, mais aussi le fonctionnement du système de fichier **sysfs**.

### Question 1

À l'aide de la commande **lsmod** vérifiez que le module *helloWord* est bien chargé dans le noyau.

### Question 2

Sachant qu'un **lsmod** parcourt la liste des modules, implémentez un module *hideModule* qui reste invisible à cette commande.

### Question 3

S'il reste caché pour la commande **lsmod**, votre module laisse encore des traces dans le système de fichier **sysfs**. En effet, au chargement d'un module l'ensemble de ses paramètres, ainsi qu'un certain nombre d'informations, sont enregistrés dans des fichiers du répertoire */sys/modules/xxx*.

Commencez par vérifier que votre module est bien visible.

### Question 4

Le **sysfs** est un système fichier spécial semblable à */proc*. Il a été introduit pour le compléter (voir le remplacer). Mais contrairement au */proc*, il présente une vue hiérarchique des informations : à chaque niveau correspond un objet : *subsystem*, *kset*, *kobject*, *attributs*.

Ces objets ne correspondent pas à des données stockées sur disque. Ils ne sont pas non plus complètement virtualisés, i.e., recalculés à chaque lecture. En fait, l'ensemble des données du **sysfs** est stocké dans la mémoire du noyau. Effacer une donnée du **sysfs** revient donc à la déréférencer puis à libérer l'espace mémoire correspondant.

Ajoutez une fonction qui efface les traces du module dans le **sysfs**, sachant que les références du **kobject** correspondant à un module, ainsi que celles de ses **attributs**, sont enregistrées dans la structure *module*.