TELECOM Paris

IP PARIS

Votre premier module

M2 SETI B4 / MS SE SE758

Guillaume Duc
guillaume.duc@telecom-paris.fr
2020–2021

Institut Mines-Télécom

Un premier module

- Nous allons rajouter du code dans le noyau pour lui ajouter des fonctionnalités (en l'occurrence, le support d'un nouveau périphérique)
- Ce code peut être ajouté
  - Soit directement parmi les autres fichiers sources dans l'arbre des sources du noyau ; la fonctionnalité peut alors être compilée *en dur* (présente en permanence) dans le noyau ou sous forme de *module* (chargeable dynamiquement)
  - Soit à l'extérieur des sources du noyau. Notre code sera alors compilé sous forme de module chargeable dynamiquement

Votre premier module

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

static int __init first_init(void)
{
  pr_info("Hello world!\n");
  return 0;
}

static void __exit first_exit(void)
{
  pr_info("Bye\n");
}

module_init(first_init);
module_exit(first_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("My first module");
MODULE_AUTHOR("Me");
```

Entêtes

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
```

- Ces entêtes contiennent la déclaration des différentes fonctions et macros utilisées : `pr_info`, `__init`, `module_init()`...
- Les modules n'utilisent pas la bibliothèque standard donc n'inclut pas les entêtes « classiques » : `stdlib.h`, `stdio.h`...

Fonction d'initialisation

```
static int __init first_init(void)
{
  pr_info("Hello world!\n");
  return 0;
}
module_init(first_init);
```

- Fonction appelée, par le noyau, lors du chargement du module

Fonction d'initialisation

```
static int __init first_init(void)
{
  pr_info("Hello world!\n");
  return 0;
}
module_init(first_init);
```

- Fonction appelée, par le noyau, lors du chargement du module
- Elle a la charge d'initialiser le module
- Valeur de retour : 0 si bien déroulée, < 0 en cas d'erreur (le module n'est alors pas chargé)

Fonction d'initialisation

```
static int __init first_init(void)
{
  pr_info("Hello world!\n");
  return 0;
}
module_init(first_init);
```

- Fonction appelée, par le noyau, lors du chargement du module
- Elle a la charge d'initialiser le module
- Valeur de retour : `0` si bien déroulée, < 0 en cas d'erreur (le module n'est alors pas chargé)
- Le nom de cette fonction est arbitraire, la macro `module_init` permet de déclarer cette fonction

Fonction d'initialisation

```
static int __init first_init(void)
{
  pr_info("Hello world!\n");
  return 0;
}
module_init(first_init);
```

- Fonction appelée, par le noyau, lors du chargement du module
- Elle a la charge d'initialiser le module
- Valeur de retour : `0` si bien déroulée, < 0 en cas d'erreur (le module n'est alors pas chargé)
- Le nom de cette fonction est arbitraire, la macro `module_init` permet de déclarer cette fonction
- Le décorateur `__init` indique que le code de la fonction peut être supprimé une fois le module chargé

Fonction d'initialisation

```
static int __init first_init(void)
{
  pr_info("Hello world!\n");
  return 0;
}
module_init(first_init);
```

- Fonction appelée, par le noyau, lors du chargement du module
- Elle a la charge d'initialiser le module
- Valeur de retour : `0` si bien déroulée, < 0 en cas d'erreur (le module n'est alors pas chargé)
- Le nom de cette fonction est arbitraire, la macro `module_init` permet de déclarer cette fonction
- Le décorateur `__init` indique que le code de la fonction peut être supprimé une fois le module chargé

Fonction de nettoyage

```
static void __exit first_exit(void)
{
  pr_info("Bye\n");
}
module_exit(first_exit);
```

- Fonction appelée, par le noyau, lors du déchargement du module
- Elle a pour objectif de désallouer les ressources utilisées par le module

Fonction de nettoyage

```
static void __exit first_exit(void)
{
  pr_info("Bye\n");
}
module_exit(first_exit);
```

- Fonction appelée, par le noyau, lors du déchargement du module
- Elle a pour objectif de désallouer les ressources utilisées par le module
- La macro `module_exit` permet de déclarer cette fonction

Fonction de nettoyage

```
static void __exit first_exit(void)
{
  pr_info("Bye\n");
}
module_exit(first_exit);
```

- Fonction appelée, par le noyau, lors du déchargement du module
- Elle a pour objectif de désallouer les ressources utilisées par le module
- La macro `module_exit` permet de déclarer cette fonction
- Le décorateur `__exit` indique que le code de la fonction peut ne pas être chargé s'il est compilé en dur ou si le déchargement des modules est désactivé

Fonction de nettoyage

```
static void __exit first_exit(void)
{
  pr_info("Bye\n");
}
module_exit(first_exit);
```

- Fonction appelée, par le noyau, lors du déchargement du module
- Elle a pour objectif de désallouer les ressources utilisées par le module
- La macro `module_exit` permet de déclarer cette fonction
- Le décorateur `__exit` indique que le code de la fonction peut ne pas être chargé s'il est compilé en dur ou si le déchargement des modules est désactivé

Méta données

```
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("My first module");
MODULE_AUTHOR("Me");
```

- `MODULE_AUTHOR` : nom et mail de l'auteur
- `MODULE_DESCRIPTION` : courte description du module
- `MODULE_LICENSE` : licence d'utilisation du module
  - Note : si la licence choisie n'est pas `GPL` ou autre+`GPL`, le module n'aura pas accès à certaines fonctions du noyau

Remarques importantes

- Il n'y a pas de fonction `main` dans un module
- Un module ne s'exécute pas « en permanence », comme le fait un programme normal
- Il n'est, la plupart du temps, qu'une collection de fonctions qui sont appelées par le noyau suite à certains événements (chargement du module, détection d'un périphérique, interruption, etc.)

Le Makefile

```
ifneq ($(KERNELRELEASE),)
# kbuild part of makefile
obj-m  := first.o

else
# normal makefile
KDIR ?= /lib/modules/`uname -r`/build

default:
	$(MAKE) -C $(KDIR) M=$$PWD
endif
```

Le Makefile

Fonctionnement : 1re passe

```
ifneq ($(KERNELRELEASE),)
    ...
else
# normal makefile
KDIR ?= /lib/modules/`uname -r`/build

default:
        $(MAKE) -C $(KDIR) M=$$PWD
endif
```

- Lors de l'appel à `make`, la variable `KERNELRELEASE` n'est normalement pas définie

Le Makefile

Fonctionnement : 1re passe

```
ifneq ($(KERNELRELEASE),)
    ...
else
# normal makefile
KDIR ?= /lib/modules/`uname -r`/build

default:
        $(MAKE) -C $(KDIR) M=$$PWD
endif
```

- Lors de l'appel à `make`, la variable `KERNELRELEASE` n'est normalement pas définie
- Appel à `make` dans le répertoire `KDIR` (sources du noyau) en positionnant la variable `M` à la valeur du chemin courant

Le Makefile

Fonctionnement : 1re passe

```
ifneq ($(KERNELRELEASE),)
   ...
else
# normal makefile
KDIR ?= /lib/modules/`uname -r`/build

default:
        $(MAKE) -C $(KDIR) M=$$PWD
endif
```

- Lors de l'appel à `make`, la variable `KERNELRELEASE` n'est normalement pas définie
- Appel à `make` dans le répertoire `KDIR` (sources du noyau) en positionnant la variable `M` à la valeur du chemin courant

Le Makefile

Fonctionnement : 1re passe

```
ifneq ($(KERNELRELEASE),)
   ...
else
# normal makefile
KDIR ?= /lib/modules/`uname -r`/build

default:
        $(MAKE) -C $(KDIR) M=$$PWD
endif
```

- Lors de l'appel à `make`, la variable `KERNELRELEASE` n'est normalement pas définie
- Appel à `make` dans le répertoire `KDIR` (sources du noyau) en positionnant la variable `M` à la valeur du chemin courant
- Le système de construction du noyau va alors rappeler notre Makefile

Le Makefile

Fonctionnement : 2e passe

```
ifneq ($(KERNELRELEASE),)
# kbuild part of makefile
obj-m  := first.o
else
   ...
endif
```

- Cette fois-ci la variable KERNELRELEASE est définie

Le Makefile

Fonctionnement : 2e passe

```
ifneq ($(KERNELRELEASE),)
# kbuild part of makefile
obj-m  := first.o
else
    ...
endif
```

- Cette fois-ci la variable `KERNELRELEASE` est définie
- Notre Makefile se contente de définir la variable `obj-m` à la valeur `first.o`

Le Makefile

Fonctionnement : 2e passe

```
ifneq ($(KERNELRELEASE),)
# kbuild part of makefile
obj-m  := first.o
else
   ...
endif
```

- Cette fois-ci la variable `KERNELRELEASE` est définie
- Notre Makefile se contente de définir la variable `obj-m` à la valeur `first.o`
- Le système de construction sait alors qu'il doit construire `first.o` (en partant de `first.c`) et le transformer en module (`first.ko`)

À vous !

- Compilez et testez ce premier module