Implémentation d'un périphérique caractère sous Linux

Gazette Linux n°125 — Avril 2006

Ranjeet Mishra

Article paru dans le n°125 de la Gazette Linux d'avril 2006.
Traduction française par Joëlle Cornavin <jcornavi CHEZ club TIRET internet POINT fr>.
Relecture de la traduction française par Encolpe Degoute <encolpe POINT degoute CHEZ free POINT fr>.

Table des matières

1. Périphérique

Pour les besoins de cet article, considérons qu'un périphérique est une représentation virtuelle, dans Linux, du matériel que l'on aimerait piloter à l'aide d'un élément de logiciel. Dans le monde de Linux, les périphériques sont implémentés sous la forme de modules. Ces derniers nous permettent de fournir la fonctionnalité de périphérique à laquelle on peut accéder depuis l'espace utilisateur.

Un point d'entrée de l'espace utilisateur vers un périphérique est fourni pas un nœud de fichier dans le répertoire `/dev`. Comme nous le savons, la plupart des éléments dans le monde Linux sont représentées sous forme de répertoires. Nous pouvons faire un **ls -l** sur n'importe quel fichier de périphérique pour connaître le type du périphérique — périphérique caractère ou bloc -, ainsi que son numéro majeur et son numéro mineur.

Le type de périphérique indique la manière dont les données sont écrites sur un périphérique. Pour un périphérique caractère, on parle d'écriture en série, octet par octet, alors que pour un périphérique bloc (par exemple, un disque dur), elle s'effectue sous forme de blocs d'octet — comme son nom le suggère.

Le numéro majeur est affecté au moment de la déclaration du périphérique (à l'aide d'un module) et le noyau l'utilise pour faire la différence entre divers périphériques. Le numéro mineur sert à l'auteur de pilotes de périphériques pour accéder à dfférentes fonctions dans le même périphérique.

Si l'on observe le nombre de fichiers dans le répertoire `/dev`, on pourrait penser qu'un très grand nombre de périphériques sont sous tension et actifs dans le système, mais en fait seuls quelques-uns pourraient être présents et actifs. Pour le vérifier, il suffit d'exécuter **cat `/proc/devices`**. (On peut alors voir les numéros majeurs et les noms des périphériques passés lors de la déclaration.)

2. Modules

Chaque périphérique nécessite un module. Des informations sur les modules actuellement chargés peuvent être extraites du noyau par le biais de **cat `/proc/modules`**. Un module n'est rien d'autre qu'un fichier objet que l'on peut lier dans un noyau actif. Pour ce faire, Linux fournit l'utilitaire insmod. À titre d'exemple, supposons que le fichier objet ***`mon_module`*** est appelé ***`mon_peripherique.o`*** : nous pouvons le lier au noyau à l'aide de **insmod *`mon_peripherique.o`***. If insmod réussit, nous pouvons voir la ligne de notre module en saisissant **cat `/proc/modules`** ou **lsmod**. On supprime le module via l'utilitaire rmmod, qui prend le nom du fichier objet comme argument.

3. Écriture d'un module pour déclarer un périphérique caractère

Avant tout, il faut connaître les bases de la génération d'un fichier objet de module. Le module utilise les fonctions de l'espace du noyau et, comme la totalité du code du noyau est écrite à l'intérieur de la directive `__KERNEL__`, nous devons la définir au moment de la compilation ou dans notre code source. On définit la directive `MODULE` avant tout, car les fonctions `MODULE` sont elles-mêmes définies à l'intérieur de celle-ci. Pour pouvoir lier notre module avec le noyau, la version du noyau actif devra correspondre à la version avec laquelle le module est compilé, sinon insmod rejettera la requête. Cela signifie que nous devons inclure le répertoire `include` présent dans le code source de Linux de la version appropriée. Là aussi, si mon module est appelé ***`mon_peripherique.c`***, un exemple d'instruction de compilateur pourrait être **gcc -D__KERNEL__ -I/usr/src/linux.2.6.7/linux/include -c *`mon_peripherique.c`***. Un paramètre `-D` sert à définir un symbole de directive quelconque. Ici, il nous faut définir `__KERNEL__` puisque sans cela, ce contenu propre au noyau ne nous sera pas accessible.

Les deux fonctions de base pour les opérations sur les modules sont `module_init` et `module_exit`. L'utilitaire insmod charge le module, appelle la fonction passée à `module_init` puis rmmod supprime le module et appelle la fonction passée à `module_exit`. Ainsi, à l'intérieur de `module_init`, nous pouvons faire ce que bon nous semble de notre API de noyau. Pour déclarer le périphérique caractère, le noyau fournit `register_chrdev` qui prend trois arguments, à savoir : le numéro majeur, la chaîne de caractères (qui donne un nom de balise au périphérique), ainsi que le pointeur `struct file_operations` qui définit tout ce que nous aimerions faire avec notre périphérique caractère. `struct file_operations` est défini dans `$(KERNELDIR)/linux/include/fs.h` qui déclare les

pointeurs de fonctions pour des opérations de base comme `open`, `read`, `write`, `release`, etc., car il faut implémenter toute fonction nécessaire pour le périphérique. Pour finir, à l'intérieur de la fonction passée à `module_exit`, nous devrons libérer les ressources à l'aide de `unregister_chrdev` qui est appelée quand nous utilisons **rmmod**.

Voici le code répertoriant l'endroit où le périphérique n'est rien d'autre qu'un bloc de 80 octets de mémoire.

4. Listings des programmes

- Makefile (outils/lg125-B/Makefile.txt)
- mon_peripherique.c (outils/lg125-B/mon_peripherique.c)
- mon_peripherique.h (outils/lg125-B/mon_peripherique.h)

5. Manipulation du périphérique caractère

Chargez le périphérique à l'aide de **insmod *`mon_peripherique.o`***, puis cherchez la ligne dans `/proc/modules` et `/proc/devices`. Créez un nœud de fichier dans le répertoire `/dev` en saisissant **mknod `/dev/mon_peripherique` c 222 0**. Examinez le code, nous lui avons donné comme numéro majeur 222. Vous pourriez penser que ce numéro risque de poser problème avec un autre périphérique — tout est correct, mais j'ai vérifié si ce numéro est déjà occupé par un autre périphérique. Il serait possible d'employer l'allocation dynamique du numéro majeur : pour ce faire, nous devons passer `0` comme argument. Lisons maintenant les données dans le périphérique en saisissant **cat `/dev/mon_peripherique`** et écrivons dans notre périphérique via **echo *`quelque_chose`* > `/dev/`*`mon_peripherique`***. Nous pouvons également écrire du code complet d'espace utilisateur pour accéder à notre périphérique à l'aide des appels système standard de `open`, `read`, `write`, `close`, etc. Un extrait du code est présenté ci-dessous.

6. Conclusion

> Vous pouvez télécharger ici (outils/lg125-B/mon_peripherique.tgz) une archive (*tarball*) contenant la totalité du code de cet article.

Dans cet article, j'ai essayé de montrer comment utiliser les fonctions du noyau pour déclarer un périphérique caractère et comment l'invoquer depuis l'espace utilisateur. Il reste beaucoup d'aspects que je n'ai pu aborder ici, tels que le problème de simultanéité, où nous devons fournir un sémaphore pour le périphérique afin d'obtenir l'exclusion mutuelle lorsque plus d'un processus est susceptible d'y accéder. J'essaierai de traiter ces problèmes dans mes futurs articles.

Je suis de New Delhi (Inde). Grand fan de Linux, j'aime la liberté que donne ce système d'exploitation pour contrôler le matériel. J'utilise Linux depuis le début du nouveau millénaire mais je n'ai commencé à creuser les

sources du noyau que récemment, après avoir achevé un *B-Tech* de l'*Indian Institute of Technology* de Guwahati. Il m'est venu l'envie de créer des modules pour commander les périphériques et je ne suis pas revenu en arrière depuis lors.

Je souhaiterais partager mes expériences et tout élément intéressant que je rencontre pendant cette aventure Linux au travers des articles de la *Linux Gazette*.