LinuxFocus article number 264
http://linuxfocus.org

par Dr. B. Thangaraju
<balasubramanian.thangaraju(at)wipro.com>

Allocation de port à sécurité intégrée pour pilotes de périphériques Linux

Résumé:

Ecrire un pilote de périphérique est autant une tâche aventureuse qu'un défi. Une fois qu'un périphérique est enregistré dans la routine "init_module" d'un pilote, les ressources système doivent lui être allouées. Le port d'entrée/sortie est l'une des ressources principales affectée au périphérique. Pour les périphériques dynamiquement liés, le développeur devra faire attention car il s'agit, dans ce cas, d'allouer une plage d'adresses de ports inutilisés. Ainsi la première tâche d'un pilote sera de vérifier le nombre de ports actifs ou libres, pour ensuite allouer uniquement des ports libres au périphérique. Enfin lorsque le module de gestion du périphérique sera supprimé du noyau, les ports devront être libérés. Cet article étudie les méandres de l'allocation de port à sécurité intégrée pour les pilotes de périphériques de Linux.

L´auteur:
Dr. B. Thangaraju a obtenu son doctorat de Physique à l'université de Bharathidasan,Tamil Nadu. Pendant cinq ans, il a été chercheur associé à l'Indian Institute of Science. Ses domaines de recherche ont été principalement : "Transparent and Conducting Oxide (TCO) thin films", "Spray Pyrolysis", "Photo Acoustic Techniques" et "p- to n-transition of Chalcogenide Glasses". Il a publié dix articles concernant ses recherches dans des journaux de renommée internationale. Il a également présenté les résultats de ses recherches dans plus de sept conférences nationales et internationales.

Il travaille à présent comme "Manager in Talent Transformation" chez Wipro Technologies en Inde. Actuellement ses recherches pour l'étude et le partage des connaissances portent sur le noyau Linux, les pilotes de périphériques, et Linux temps réel.

Traduit en Français par:

Guy Passemard <g.passemard(at)free.fr>

Introduction

Le principal souci du développeur de pilote de périphérique est l'allocation des ressources. Ces ressources sont les ports d'entrée/sortie, la mémoire et la gestion des interruptions (IRQ). Cet article essaie d'expliquer les notions fondamentales du sous-système de gestion des entrées/sorties et l'importance de l'allocation des ressources aux ports d'entrées/sorties. Cet article distingue clairement les étapes de contrôles, d'affectation et de libération des adresses de ports de périphériques.

Les élements matériel de base d'un ordinateur, tels que les ports, les bus et les contrôleurs, permettent de connecter la plupart des périphériques d'E/S. Les pilotes de périphériques offrent un accès uniformisé au sous-système d'E/S, comparable aux appels système qui proposent une interface standard entre les applications et le système d'exploitation. Ainsi de nombreux périphériques sont reliés à l'ordinateur : par exemple, les dispositifs de stockage tels que les unités de disques, de bandes, les CD-ROM, et les disquettes, ou les interfaces avec l'utilisateur comme le clavier, la souris et l'écran, sans oublier le matériel de communication comme les cartes réseau et les modems. Malgré la variété des périphériques proposés, nous avons seulement besoin de comprendre quelques principes généraux sur leur connexion et sur la manière dont le logiciel contrôle le matériel.

Concepts Fondamentaux

Un périphérique est composé de deux parties : l'une électronique appelée contrôleur de périphérique, et l'autre qui est mécanique. Le contrôleur communique avec l'ordinateur à travers le bus système. Habituellement, un groupe d'adresses de ports (sans conflit) est affecté à chaque contrôleur. Les ports d'entrée/sortie utilisent quatre jeux de registres nommés "status", "control", "data-in", "data-out". A la lecture des bits contenus dans le registre "status", l'ordinateur sait : si la commande en cours est terminée, si un octet est prêt à être lu ou écrit, ou s'il y a erreur. Le registre "control" est mis à jour par l'hôte lors du lancement d'une commande ou lors du changement de mode du périphérique. Le système peut accéder aux données grâce au registre "data-in" pour les entrées et inversement en sortie avec le registre "data-out".

Donc, l'interface de base, entre le processeur et le périphérique, est un ensemble de registres de contrôle et d'états. Quand le processeur exécute un programme et qu'il rencontre une instruction relative à un périphérique, il exécute l'instruction en lançant une commande destinée au périphérique approprié. Le contrôleur traite l'action demandée et positionne les bits correspondants dans le registre "status" et attend. C'est le processeur qui doit vérifier l'état du périphérique jusqu'à la fin de l'opération. Par exemple, le pilote du port parallèle (utilisé par une imprimante) va périodiquement scruter l'état de l'imprimante pour vérifier qu'elle est prête à recevoir des données sinon il se mettra en attente laissant le processeur libre pour effectuer d'autres tâches et recommencera jusqu'à ce que l'imprimante soit prête. Ce mécanisme de scrutation améliore les performances et évite au système d'attendre inutilement l'activation du périphérique sans rien faire d'autre.

Les registres ont des adresses fixes définies dans l'espace d'E/S. Généralement celles-ci sont affectées

au démarrage du système, en fonction des paramètres définis dans un fichier de configuration d'installation du système, ainsi une plage d'adresses peut être allouée à chaque périphérique statique. Le noyau contient donc les pilotes des périphériques présents et les plages d'adresses E/S de ports d'E/S peuvent être stockées dans le répertoire **proc**. Vous pouvez vérifier les plages d'adresses de ports des périphériques présents sur votre système avec la commande **$cat /proc/interrupts**. La première colonne affichée indique la plage d'adresses des ports et la seconde le périphérique propriétaire des ports. Certains OS sont capables de charger dynamiquement les pilotes en cours de fonctionnement du système. Ainsi tout nouveau périphérique peut être reconnu par le système et contrôlé ou accédé par le module du pilote dynamiquement chargé.

Le concept de pilote de périphérique est relativement abstrait, et il se situe au plus bas niveau du logiciel fonctionnant sur un ordinateur, puisqu'il est directement lié aux caractéristiques matérielles du périphérique . Un pilote ne gère qu'un seul type de périphérique. Ce type peut être caractère, bloc ou réseau. Si une application appelle un périphérique, le noyau contacte le pilote de ce périphérique. Le pilote envoie alors une commande à ce périphérique. Le pilote n'est qu'une collection de fonctions avec de nombreux points d'entrée comme open, close, read, write, ioctl, llseek, etc. Quand vous insérer un module la fonction "init_module()" est appelée et lorsque le module est retiré c'est la fonction "cleanup_module()" qui est executée. Le périphérique est enregistré dans un pilote de la routine "init_module".

Un fois le périphérique enregistré dans init_module(), ses ressources telles que les ports E/S, la mémoire et les niveaux d'IRQ sont alloués dans la fonction même, afin de permettre un fonctionnement correct du pilote. Si une adresse erronée est allouée, le noyau envoie un message **segmentation fault**. Par contre, dans le cas des ports E/S, le noyau n'enverra pas de message comme **wrong I/O port**. Pourtant, assigner des ports déjà utilisés par des périphériques existants fera "tomber" le système. Lorsque vous retirez le module, le périphérique ne doit plus être enregistré, c'est-à-dire que le nombre majeur redevient disponible et que les ressources sont libérées dans la fonction clean_up module ( ).

Le travail le plus fréquemment executé par un pilote de périphérique est la lecture et l'écriture sur les ports d'E/S. Ainsi il faut s'assurer que pour le pilote, l'usage des adresses de ports soit exclusif. Aucun autre périphérique ne doit utiliser cette plage d'adresses. Pour en être certain le premier pilote devra vérifier si l'adresse est disponible ou non. Lorsque le pilote constate que l'adresse est inutilisée, il demande au noyau d'assigner la plage d'adresses à son périphérique.

Allocation de port à sécurité intégrée

Maintenant nous allons voir comment allouer et libérer les ressources affectées aux périphériques par les fonctions du noyau. L'approche pratique suivante est décrite pour un noyau Linux 2.4. Ainsi ce qui suit n'est totalement applicable qu'à Linux, et dans une certaine mesure à des variantes d'UNIX.

Premièrement, il s'agit de contrôler la disponibilité des plages d'adresses de ports avec la fonction :

int check_region (unsigned long start, unsigned long len);

La fonction renvoie zéro si la plage d'adresses est disponible, sinon un nombre négatif ou un code erreur

(-EBUSY ou -EINVAL). La fonction accepte deux arguments : **start** est l'adresse de départ d'une plage de ports, **len** est le nombre de ports dans la plage.

Une fois le port disponible, il doit être alloué au périphérique par la fonction request_region.

struct resource *request_region (unsigned long start, unsigned long len, char *name);

Les deux premiers arguments sont les mêmes que ceux vus précédemment, le pointeur de la variable caractère **name** sert à définir le nom du périphérique auquel l'adresse du port est allouée. La fonction renvoie le type de pointeur à la structure "resource". Cette structure permet de décrire l'étendue des ressources et elle est déclarée dans le fichier <linux/ioport.h>. Voici la syntaxe de la structure :

```
struct resource {
        const char *name;
        unsigned long start, end;
        unsigned long flags;
        struct resource *parent, *sibiling, *child;
};
```

Lorsqu'un module est retiré du noyau, le port doit être libéré pour pouvoir être utilisé par d'autres périphériques. Pour cela il faut utiliser la fonction release_region ( ) dans cleanup_module ( ). La syntaxe de la fonction est :

void release_region (unsigned long start, unsigned long len);

Les arguments de cette fonction sont les mêmes que précédemment. Il faut savoir que ces trois fonctions sont en fait des macros déclarées dans le fichier <linux/ioport.h>.

Exemple de code d'un pilote pour allocation de port de périphérique

Le programme qui suit montre l'allocation et la déallocation de ports pour un périphérique chargé dynamiquement.

```
#include <linux/fs.h>
#include <linux/ioport.h>

struct file_operations fops;
unsigned long start, len;
int init_module (void)
{

 int status;
 start = 0xff90;
 len   = 0x90;
 register_chrdev(254,"your_device",&fops);

 status =  check_region (start, len);
 if (status == 0) {
     printk ("The ports are available in that range\n");
```

```
     request_region(start,len,"your_device");
 } else {
     printk ("The ports are already in use. Try other range.\n");
     return (status);
 }
 return 0;
}

void cleanup_module (void)
{
 release_region(start, len);
 printk ("ports are freed successfully\n");
 unregister_chrdev(254,"your_device");}
 printk (" your device is unregistered\n");
}
```

Pour éviter toute confusion, le contrôle et l'allocation dynamique du nombre majeur ont été omis. Une fois le port alloué avec succès, il est possible de vérifier dans le répertoire proc :
$cat /proc/ioports

Les fonctions du noyau : options concernant les ports d'E/S

La gestion des ports d'E/S de Linux dispose de nombreuses fonctions. Leur utilisation dépend de la place mémoire requise en lecture - écriture. Les ports peuvent être de 8, 16 ou 32 bits. Le noyau Linux dispose du fichier entête <asm/io.h> qui définit les fonctions d'E/S des ports. Les fonctions suivantes servent à la lecture (inx) et l'écriture (oux) de ports transmettant des données de 8,16 ou 32 bits :

```
__u8 inb (unsigned int port);
void outb (__u8 data, unsigned int port);

__u16 inw (unsigned int port);
void outw(__u16 data, unsigned int port);

__u32 inl (unsigned int prot);
void outl (__u32 data, unsigned int port);
```

D'autre part, pour transmettre efficacement des chaînes de caractères on peut se servir des fonctions suivantes :

```
void insb(unsigned int port, void *addr, unsigned long count);
void outsb(unsigned int port, void *addr, unsigned long count);
```

les données sont lues ou écrites sur le port dénommé "port". "addr" représente l'adresse mémoire du début de la chaîne à transférer et "count" le nombre d'unités à transférer.

void insw(unsigned int port, void *addr, unsigned long count);
void outsw(unsigned int port, void *addr, unsigned long count);

lit ou écrit des valeurs 16 bits sur un seul port 16 bits.

void insl(unsigned int port, void *addr, unsigned long count);
void outsl(unsigned int port, void *addr, unsigned long count);

lit ou écrit des valeurs 32 bits sur un seul port 32 bits.

Remerciements

L’auteur remercie **Monsieur Jayasurya V**, Manager, Talent Transformation, Wipro Technologies, India, pour sa relecture critique du manuscrit.

Références

- Linux Device Drivers (2nd Edition), par Alessandro Rubini et Jonathan Corbet. Ce livre est disponible chez l’éditeur O’Reilly: http://linux.oreilly.com
- Linux Kernel 2.4 Internals: http://tldp.org/LDP/lki/index.html
- Linux Kernel Module Programming Guide: http://tldp.org/LDP/lkmpg/mpg.html
- The Linux Kernel (older guide, 1998): http://tldp.org/LDP/tlk/tlk.html

---