Programmation système – TP5

Objectifs du TP :

1. Parcours et reconnaissance des périphériques PCI ;
2. Manipulation des caractéristiques des périphériques PCI ;
3. Traitement des interruptions.

1 Parcours et reconnaissance des périphériques PCI

On commencera par reconnaître tout ou partie des périphériques PCI disponibles. On pourra utiliser `lspci -n` pour obtenir les identifiants des périphériques de la machine.

Pour reconnaître un périphérique, on définit des `pci_device_id` où on place ses identifiants ou des `PCI_ANY_ID` lorsqu'on se fiche de la valeur.

```
struct pci_device_id {
  __u32 vendor, device;           /* Vendor and device ID or PCI_ANY_ID*/
  __u32 subvendor, subdevice;     /* Subsystem ID's or PCI_ANY_ID */
  __u32 class, class_mask;        /* (class,subclass,prog-if) triplet */
  kernel_ulong_t driver_data;     /* Data private to the driver */
};
```

La table des `pci_device_id` est placée dans une structure `pci_driver` qui est passée au système par `pci_module_init`. Le noyau parcourt la liste des périphériques (non déjà utilisés par un autre pilote) et si leurs identifiants correspondent à une entrée la table `id_table`, il réserve le périphérique et l'initialise par la méthode `probe` du `pci_driver`.

```
#include <linux/pci.h>

struct pci_driver {
  char *name;
  struct pci_device_id *id_table;
  int (*probe) (struct pci_dev *, const struct pci_device_id *);
  void (*remove) (struct pci_dev *);
};

int pci_register_driver(struct pci_driver * driver);
void pci_unregister_driver(struct pci_driver * driver);
```

2 Caractéristiques des périphériques PCI

Pour chaque périphérique reconnu, on affichera ses ressources propres à la manière de `lspci -nv`.

La fonction `probe` du pilote est lancée et reçoit un `pci_dev` qui décrit le périphérique en détail.

```
unsigned long pci_resource_start(dev, num);
unsigned long pci_resource_end(dev, num);
unsigned long pci_resource_flags(dev, num);
```

On peut notamment récuperer les différentes ressources exportées par le périphérique (début, fin, type).

```
/* type de ressource */
#define  PCI_BASE_ADDRESS_SPACE_IO     0x01
#define  PCI_BASE_ADDRESS_SPACE_MEMORY 0x00

/* type de mémoire */
#define  PCI_BASE_ADDRESS_MEM_TYPE_MASK 0x06
#define  PCI_BASE_ADDRESS_MEM_TYPE_32   0x00    /* 32 bit address */
#define  PCI_BASE_ADDRESS_MEM_TYPE_1M   0x02    /* Below 1M [obsolete] */
#define  PCI_BASE_ADDRESS_MEM_TYPE_64   0x04    /* 64 bit address */

/* mémoire préfetchable ou non */
#define  PCI_BASE_ADDRESS_MEM_PREFETCH  0x08
```

3 Interruptions

On définira ensuite un traitant d'interruption pour les périphériques concernés.

Le champ `irq` des `pci_dev` donne le numéro de ligne d'interruption.

```
#include <linux/interrupt.h>
int request_irq(unsigned int irq,
                irqreturn_t (*handler)(int, void *, struct pt_regs *),
                unsigned long flags, const char * name, void * id);
void free_irq(unsigned int irq, void * id);
```

Le traitant doit renvoyer `IRQ_NONE` s'il n'était pas concerné, et `IRQ_HANDLED` s'il l'a traitée.