Programmation système – TP6

Objectifs du TP :

1. Commande du haut-parleur interne par les ports d'entrées-sorties ;
2. Utilisation des interruption clavier pour modifier le comportement du haut-parleur ;
3. Partage de variable entre le traitant d'interruption et d'autres contextes.

1 Commande du haut-parleur

Le haut-parleur interne du PC est actif lorsque le bit 1 du port `0x61` est placé. On utilise généralement le *timer* interne pour le guider. Pour cela, on placera également le bit 0 de ce port.

Auparavant, il faudra avoir configuré ce *timer*. On le place en mode *oscillateur* en envoyant la commande `0xb6` dans le port `0x43`. On écrit sa fréquence d'oscillation (codée sur deux octets) en deux parties dans le port `0x42`. Le dernier octet y est écrit, puis le premier.

Le haut-parleur étant un périphérique ISA, il est assez lent par rapport aux machines actuelles. Il faudra donc utiliser des variantes avec pause des fonctions `inb/outb`.

```
u8 inb_p(unsigned long port);
void outb_p(u8 val, unsigned long port);
```

2 Interruptions

On définit un traitant d'interruption pour le clavier et on y modifie la fréquence du bruit émis par le hauteur parleur.

```
#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 clavier est géré par le driver `i8042`. On peut récupérer sa ligne d'interruption en regardant dans `/proc/interrupts`. En général, c'est 1. Comme on ne traite pas réellement l'interruption, on doit renvoyer `IRQ_NONE` dans le traitant.

3 Partage de variable avec le traitant d'interruption

Les fonctions d'accès aux registres des périphériques sont risquées dans un traitant d'interruption car celui-ci doit être très court.

On va donc utiliser un thread noyau pour modifier la fréquence du haut-parleur suite aux événements détectés dans le traitant d'interruption. On partagera donc une variable entre le thread et le traitant. Il faudra donc la protéger.

```
#include <linux/spinlock.h>
void spin_lock_init(spinlock_t * lock);
void spin_lock_irqsave(spinlock_t * lock, unsigned long irqflags);
void spin_unlock_irqrestore(spinlock_t * lock, unsigned long irqflags);
```