Programmation système – TP3

Objectifs du TP :

1. Lancer, réveiller et stopper un thread noyau ;
2. Utiliser un *character device* pour transférer des données entre noyau et applications.

1 Thread noyau (fin du TP2)

1.1 Lancement du thread

```
#include <linux/sched.h>
int kernel_thread(int(*func)(void *), void * arg, unsigned long flags);
```

1.2 Terminaison du thread

```
void init_completion(struct completion *);
void wait_for_completion(struct completion *);
void complete_and_exit(struct completion *, long code);
```

Le thread appelle `complete_and_exit` pour terminer. Son père attend sa terminaison par `wait_for_completion`.

1.3 Sommeil et réveil du thread

```
void init_waitqueue_head(wait_queue_head_t *);
void wait_event(wait_queue_head_t, int test);
void wake_up(wait_queue_head_t *);
```

Le thread s'endort dans la file d'attente avec `wait_event` jusqu'à ce qu'on le réveille et que `test` soit non-nul. On réveille un processus dans la file d'attente avec `wake_up`.

2 Communication user-noyau par character device

2.1 Création du character device dans le noyau

```
#include <linux/fs.h>
int register_chrdev(unsigned int major, char * name,
                    struct file_operations * fops);
void unregister_chrdev(unsigned int major, char * name);
```

On enregistre un *character device* dont les opérations de contrôle sont `fops` et le numéro de majeur `major`. Le périphérique apparaît dans `/proc/devices`.

```
struct file_operations {
  ssize_t (*read) (struct file *, char *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
  int (*open) (struct inode *, struct file *);
  int (*release) (struct inode *, struct file *);
};

static int mychrdev_open(struct inode * inode,
                         struct file * file) {
```

```
  ...
  return 0;
}

static struct file_operations mychrdev_fops = {
  .open = mychrdev_open,
};
```

On définit les opérations spécifiques, le système utilisera les opérations par défaut pour les autres. L'ouverture du device créé un `struct file*` dont le champ `private_data` peut être utilisé à loisir pour stocker des données spécifiques.

2.2 Accès au character device depuis l'espace utilisateur

```
mknod /dev/mychrdev c <major> <minor>
chmod 644 /dev/mychrdev
cat > /dev/mychrdev
cat < /dev/mychrdev
```

Le fichier spécial pointe alors vers le *character device* créé dans le noyau. On peut alors y écrire ou y lire des données.

2.3 Copie de données entre espace utilisateur et noyau

```
#include <asm/uaccess.h>
unsigned long copy_from_user(void *kto, const void * ufrom, unsigned long size);
unsigned long copy_to_user(void *uto, const void *kfrom, unsigned long size);
```

Transfert de données entre espace utilisateur (`uto` ou `ufrom`) et espace noyau (`kfrom` ou `kto`).

```
int put_user(val, void *uptr);
int get_user(val, const void *uptr);
```

C'est le type du pointeur cible en espace utilisateur (`*uptr`) qui définit la quantité de données copiées (1, 2, 4 ou 8 octets).