Université Tunis El-Manar	**Année Universitaire** : 2022-2023
Faculté des Sciences de Tunis	**Module :** Linux embarqué
Section : LIOT2	**Enseignant :** C.A. ABID

Exercice.

On se propose de développer un module noyau en mode caractère qui simule le fonctionnement d'une pile de caractères.

On se limite à gérer la pile à travers un tableau de caractères. Le fichier spécial à associer au module possède le numéro majeur **190**. Un tel fichier peut être créé à travers la commande suivante :

```
# mknod /dev/pile c 190 0
```

Ainsi, pour empiler des éléments dans la pile, on doit écrire dans le fichier spécial **/dev/pile** :

Exemple : **echo "Bonjour ma pile">/dev/pile**

Pour dépiler des éléments de la pile, il suffit de lire les données à partir du fichier **/dev/pile**.

Corrigé.

```
#include<linux/init.h>
#include<linux/module.h>
#include<linux/fs.h>
#include<linux/string.h>
#include<asm/uaccess.h>
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Pile");
static char ker_buf[100];   //driver local buffer
static int indice;
//définition des structures de gestion du FS à implémenter//
static int dev_open(struct inode *inod, struct file *fil);
static ssize_t dev_read(struct file *filep, char *buf, size_t len, loff_t *off);
static ssize_t dev_write(struct file *flip, const char *buff, size_t len,
            loff_t *off);
static int dev_release(struct inode *inod, struct file *fil);
static long notredev_ctl(struct file *filep, unsigned int cmd,
            unsigned long arg);
//structure contenant les opérations du périphérique
static struct file_operations fops = { .read = dev_read, //pointeur vers la
fonction read
            .write = dev_write, //pointeur vers la fonction write
            .open = dev_open,   //pointeur vers la fonction open
            .release = dev_release, //pointeur vers la fonction release
```

```
            .unlocked_ioctl = notredev_ctl, };

static int init(void)   { //fonction d'initialisation
      int t = register_chrdev(190, "pile", &fops); //création du périphérique
                              //caractère de majeur 190 et de nom pile
      indice = 0;
      if(t<0)
      printk(KERN_ALERT "device registration failed.");
      else
      printk(KERN_ALERT "device registred\n");
      return 0;
}

static void exit_(void) {//fonction de clôture
      unregister_chrdev(190, "pile");
      printk(KERN_ALERT "exit");
}

static int dev_open(struct inode *inod, struct file *fil) {
      printk("KERN_ALERT device opened");
      return 0;
}

static ssize_t dev_read(struct file *filep, char *buf, size_t len, loff_t *off)
{
      int i = 0;
      for (i = 0; i < len && indice > 0; i++) {
            put_user(*(ker_buf + indice - 1), buf + i);
            indice--;
      }

      return i;
}

static ssize_t dev_write(struct file *flip, const char *buf, size_t len, loff_t
*off) {
      printk("Ecriture\n");
      int i = 0;
      for(i = 0; i < len && indice < 100; i++) {
            get_user(ker_buf[indice], buf + i);
            indice++;
      }
      return len;
```

```
}

static int dev_release(struct inode *inod, struct file *fil) {
      printk("KERN_ALERT device closed\n");
      return 0;
}

static long notredev_ctl(struct file *filep, unsigned int cmd,unsigned long arg)
{
      if (cmd == 1) {
            if (arg == 1) indice = 0;
            else if (arg == 2)      return indice;
      }
      return indice;
}

module_init (init);
module_exit (exit_);
```