Travail Pratique Bonus

Modules noyau et pilote en mode caractère sous Linux

Emilien Lancelot – 111163698

Jonathan Quach - 111 163 730

GLO-2001

# Question 1)

Notez que pour tous les exercices la ligne « /usr/src/linux-headers-4.4.0-57-generic/ » a été remplacée par « /usr/src/linux-headers-4.4.0-64-generic/ » pour que cela compile.

Sortie des commandes :

$ insmod helloworld.ko

$ lsmod

Module Size Used by

helloworld 16384 0

rfcomm 65536 12

bnep 20480 2

coretemp 16384 0

[….]

$ rmmod helloworld.ko

$ dmesg

[…]

[80478.472492] Hello world!

[80632.398993] Goodbye world!

# Question 2

**2) i)**

#include <linux/module.h>

#include <linux/init.h>

#include <linux/types.h>

#include <linux/fs.h>

#include <linux/cdev.h>

struct cdev \* p\_vircdev;

dev\_t dev\_num;

int major\_number;

int minor\_number;

#define DEVICE\_NAME "my\_virtual\_device"

#define FIRST\_MINOR 0

#define NB\_MINOR\_ID 1

static int driver\_init(void)

{

int err;

err = alloc\_chrdev\_region(&dev\_num, FIRST\_MINOR, NB\_MINOR\_ID, DEVICE\_NAME);

if (err < 0) {

printk(KERN\_ALERT "%s: failed to obtain device numbers\n", DEVICE\_NAME);

return err;

}

p\_vircdev = cdev\_alloc();

p\_vircdev->owner = THIS\_MODULE;

err = cdev\_add(p\_vircdev, dev\_num, NB\_MINOR\_ID);

major\_number = MAJOR(dev\_num);

minor\_number = MINOR(dev\_num);

if (err < 0) {

printk(KERN\_ALERT "%s: unable to add cdev to kernel\n", DEVICE\_NAME);

return err;

}

printk(KERN\_ALERT "%s: module successfully loaded\n", DEVICE\_NAME);

printk(KERN\_INFO "\tuse \"mknod /dev/%s c %d %d\" for device file", DEVICE\_NAME, major\_number, minor\_number);

return 0;

}

static void driver\_cleanup(void)

{

unregister\_chrdev\_region(dev\_num, NB\_MINOR\_ID);

cdev\_del(p\_vircdev);

printk(KERN\_ALERT "Module %s successfully unloaded\n", DEVICE\_NAME);

}

module\_init(driver\_init);

module\_exit(driver\_cleanup);

**2) ii)**

$ insmod reg\_driver.ko

$ dmesg

[...]

[ 79.511247] my\_virtual\_device: module successfully loaded

[ 79.511250] use "mknod /dev/my\_virtual\_device c 246 0" for device file

$ rmmod reg\_driver

$ dmesg

[...]

[ 120.315371] Module my\_virtual\_device successfully unloaded

**2) iii)**

$ cat /proc/devices

Character devices:

[...]

226 drm

246 my\_virtual\_device

247 hidraw

248 aux

[...]

**2) iv)**

Ajout des droits à tous les utilisateurs avec chmod:

$ chmod a=rwx /dev/my\_virtual\_device

$ ls -l /dev/my\_virtual\_device

crwxrwxrwx 1 root root 246, 0 Apr 27 15:34 /dev/my\_virtual\_device

# Question 3)

**3) i)**

#include <linux/module.h>

#include <linux/init.h>

#include <linux/types.h>

#include <linux/fs.h>

#include <linux/cdev.h>

#include <asm/uaccess.h>

#include <linux/slab.h>

#include <linux/semaphore.h>

#include <linux/mutex.h>

#define DEVICE\_NAME "my\_virtual\_device"

#define FIRST\_MINOR 0

#define NB\_MINOR\_ID 1

#define DISK\_SIZE 1024\*1024

// The virtual device using RAM

struct CBuffDevice {

char data[DISK\_SIZE];

unsigned long head;

unsigned long tail;

unsigned long size;

struct semaphore sem;

} virtual\_device;

// Character driver specific information

struct cdev \* p\_vircdev;

dev\_t dev\_num;

int major\_number;

int minor\_number;

//Méthode appelée lors de l'ouverture du périphérique

int device\_open(struct inode \*inode, struct file \*filp)

{

printk(KERN\_ALERT "%s: opened device on inode %lu\n", DEVICE\_NAME, inode->i\_ino);

return 0;

}

//Méthode appelée lors de la fermeture du périphérique

int device\_release(struct inode \*inode, struct file \*filp)

{

printk(KERN\_ALERT "%s: released device on inode %lu\n", DEVICE\_NAME, inode->i\_ino);

return 0;

}

//Méthode appelée lors d'une opération d'écriture sur le périphérique

ssize\_t device\_write(struct file\* filp, const char\* bufUserData, size\_t count, loff\_t\* f\_pos)

{

int i;

int j;

int nb;

char \*tampon;

//Verification que l'on peut écrire au moins un caractère

if (virtual\_device.size == DISK\_SIZE)

return -ENOSPC;

i = 0;

j = virtual\_device.tail;

nb = 0;

//Allocation du tampon pour copy\_from\_user

tampon = kmalloc(count \* sizeof(char), GFP\_KERNEL);

if (copy\_from\_user(tampon, bufUserData, count) != 0)

{

return -EFAULT;

}

//Debut de la section critique

down\_interruptible(&virtual\_device.sem);

while (i < count)

{

//Copie du buffer dans le disque

virtual\_device.data[j] = tampon[i];

i++;

j++;

nb++;

//Fin du disque. Retour au debut (buffer circulaire)

if (j > DISK\_SIZE - 1)

{

virtual\_device.tail = 0;

j = 0;

}

//Plus de place sur le disque

if ((virtual\_device.size + nb) == DISK\_SIZE)

break;

}

//Fin de la section critique

up(&virtual\_device.sem);

virtual\_device.tail = j;

virtual\_device.size += nb;

(\*f\_pos) += nb;

printk(KERN\_ALERT "%s: head:%lu tail:%lu size:%lu\n", DEVICE\_NAME, virtual\_device.head, virtual\_device.tail, virtual\_device.size);

return nb;

}

//Méthode appelée lors d'une opération de lecture sur le périphérique

ssize\_t device\_read(struct file\* filp, char\* bufUserData, size\_t count, loff\_t\* f\_pos)

{

int nb;

int i;

int j;

char \*kStr;

nb = 0;

i = 0;

j = virtual\_device.head;

//Allocation du tampon pour copy\_to\_user

kStr = kmalloc((count \* sizeof(char)), GFP\_KERNEL);

//Debut de la zone critique

down\_interruptible(&virtual\_device.sem);

while (i < count && i < virtual\_device.size)

{

//Copie dans le tampon

kStr[i] = virtual\_device.data[j];

virtual\_device.data[j] = '\0';

i++;

j++;

nb++;

//Fin du disue retour au debut (buffer circulaire)

if (j > DISK\_SIZE - 1)

{

virtual\_device.head = 0;

j = 0;

}

}

//Fin de la zone critique

up(&virtual\_device.sem);

virtual\_device.head = j;

virtual\_device.size -= nb;

if (copy\_to\_user(bufUserData, kStr, nb) != 0)

{

return -EFAULT;

}

(\*f\_pos) += nb;

return nb;

}

// Specifying functions responding to file system calls

struct file\_operations fops = {

.owner = THIS\_MODULE,

.open = NULL,

.release = NULL,

.write = NULL,

.read = NULL,

};

//méthode appelée lors de l'initialisation du pilote

static int driver\_init(void)

{

int err;

int i;

//Routage des fonctions

fops.open = device\_open;

fops.release = device\_release;

fops.write = device\_write;

fops.read = device\_read;

sema\_init(&virtual\_device.sem, 1);

//Allocation du périphérique

err = alloc\_chrdev\_region(&dev\_num, FIRST\_MINOR, NB\_MINOR\_ID, DEVICE\_NAME);

if (err < 0) {

printk(KERN\_ALERT "%s: failed to obtain device numbers\n", DEVICE\_NAME);

return err;

}

p\_vircdev = cdev\_alloc();

p\_vircdev->owner = THIS\_MODULE;

p\_vircdev->ops = &fops;

err = cdev\_add(p\_vircdev, dev\_num, NB\_MINOR\_ID);

if (err < 0) {

printk(KERN\_ALERT "%s: unable to add cdev to kernel\n", DEVICE\_NAME);

return err;

}

//init le buffer circulaire

i = 0;

while (i < DISK\_SIZE)

{

virtual\_device.data[i] = '\0';

i++;

}

virtual\_device.head = 0;

virtual\_device.tail = 0;

virtual\_device.size = 0;

//Recuperation du minor et major

major\_number = MAJOR(dev\_num);

minor\_number = MINOR(dev\_num);

printk(KERN\_ALERT "%s: module successfully loaded\n", DEVICE\_NAME);

printk(KERN\_INFO "\tuse \"mknod /dev/%s c %d %d\" for device file", DEVICE\_NAME, major\_number, minor\_number);

return 0;

}

//Méthode appelée lors de la suppression du périphérique

static void driver\_cleanup(void)

{

unregister\_chrdev\_region(dev\_num, NB\_MINOR\_ID);

cdev\_del(p\_vircdev);

printk(KERN\_ALERT "Module %s successfully unloaded\n", DEVICE\_NAME);

}

module\_init(driver\_init);

module\_exit(driver\_cleanup);

**3) ii)**

Fonction d’écriture :

Dans un premier temps nous vérifions qu’il reste de la place sur le disque en comparant l’attribut size de notre virtual\_device avec la taille du disque. S’il ne reste plus de place alors on renvoie le code -ENOSPC.

Ensuite on alloue un char \* tampon de la taille de la variable count (passée en paramètre). On utilisera cette variable dans la fonction copy\_from\_user() qui va la remplir avec les données qui seront à écrire sur le disque virtuel.

Enfin on rentre dans la section critique avec un semaphore. Dans cette section nous copions caractère par caractère le contenu du tampon dans le disque virtuel. Si l’on arrive à la fin du disque on remet notre « tail » à 0 et la copie reprend jusqu’à ce que tout soit écrit ou qu’il n’y y ait plus de place sur le disque. Une fois tous les caractères copiés on sort de la zone critique. Il ne reste plus qu’à maintenir à jour notre « tail » du buffer circulaire et la taille du contenu du disque. Pour finir on renvoie le nombre de caractères écrit.

Fonction de lecture :

Dans un premier temps on alloue un char \* tampon de la taille de count (passé en paramètre).

Ensuite on rentre dans une section critique grâce à un semaphore. Dans cette section critique nous copions caractère par caractère l’intégralité du contenu du disque dans la variable tampon. Si l’on arrive à la fin du disque on remet le “head” du buffer circulaire à 0 et la copie reprend. Une fois terminé on sort de la zone critique.

Enfin le tampon ayant le contenu du disque virtuel est donné à la méthode copy\_to\_user() qui glissera le contenu du tampon vers la variable « bufUserData » passée en paramètre et qui sera retournée à l’utilisateur par son pointeur. Enfin on renvoie le nombre de caractère lus lors de l’opération.

**3) iii)**

echo bonjour > /dev/my\_virtual\_device

cat /dev/my\_virtual\_device

bonjour

dmesg

[ 73.851580] my\_virtual\_device: module successfully loaded

[ 73.851584] use "mknod /dev/my\_virtual\_device c 246 0" for device file

[ 258.393154] my\_virtual\_device: opened device on inode 464

[ 258.393171] my\_virtual\_device: head:0 tail:8 size:8

[ 258.393175] my\_virtual\_device: released device on inode 464

[ 371.172288] my\_virtual\_device: opened device on inode 464

[ 371.172315] my\_virtual\_device: released device on inode 464