Martin Zavalla Gamarra

Trabajo Practico 0: Device Drivers

1) Usando el código provisto y los comandos de buildeo y carga/descarga de drivers

Pre-requisitos: instalar Linux headers, la herramienta make y build-essential(GCC)
Make clean -> limpia los archivos
Make -> crea los archivos necesarios para cargar un driver
Insmod nombreDeMoudlo.ko -> carga un driver en el kernel
Rmmod nombreDeMoudlo –> descarga un driver del kernel

```
alumno@alumno-virtualbox:~/sor2/TP0$ ls
Makefile  reverseDriver.c  script.sh
alumno@alumno-virtualbox:~/sor2/TP0$ make clean && make
make -C /lib/modules/5.4.0-144-generic/build M=/home/alumno/sor2/TP0 clean
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-144-generic'
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-144-generic'
make -C /lib/modules/5.4.0-144-generic/build M=/home/alumno/sor2/TP0 modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-144-generic'
  CC [M]  /home/alumno/sor2/TP0/reverseDriver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/alumno/sor2/TP0/reverseDriver.mod.o
  LD [M]  /home/alumno/sor2/TP0/reverseDriver.ko
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-144-generic'
alumno@alumno-virtualbox:~/sor2/TP0$ ls
Makefile       Module.symvers   reverseDriver.ko   reverseDriver.mod.c  reverseDriver.o
modules.order  reverseDriver.c  reverseDriver.mod  reverseDriver.mod.o  script.sh
alumno@alumno-virtualbox:~/sor2/TP0$ sudo insmod reverseDriver.ko
[sudo] contraseña para alumno:
alumno@alumno-virtualbox:~/sor2/TP0$ sudo rmmod reverseDriver
```

2) Verificar que funcione. ¿Cómo hacemos eso?

El comando dmesg, nos permite ver el log del kernel. En este caso, cuando usamos **insmod**, se imprime un mensaje de registro con las instrucciones a seguir para comunicarse con el driver y cuando usamos el **rmmod** imprime un mensaje de des registro

```
[ 5205.298826] UNGS : Driver registrado
[ 5205.298831] I was assigned major number 240.To talk to the driver create a dev file with
[ 5205.298833] mknod -m 666 /dev/device_reverso c 240 0
[ 5205.298833] Try to cat and echo to the device file.
[ 5205.298834] Remove the device file and module when done.
[ 5251.287576] UNGS : Driver desregistrado
```

3) Definir funciones *init_module* y *cleanup_module*

El método register_chrdev() sirve para cargar un driver y recibe tres parámetros.
Major number -> se puede elegir manualmente un número que no esté en uso por otro driver o colocando un 0 para que el valor de retorno sea un numero asignado dinámicamente.
Nombre -> nombre del driver
File operations -> puntero a operaciones de archivos

Printk es una función usamos para imprimir mensajes al log del kernel para brindar información sobre el driver y como comunicarse con él.

El método unregister_chrdev() sirve para descargar un driver y recibe dos parámetros. El major number y el nombre del driver.

```
101 int init_module(void)
102 {
103   Major = register_chrdev(0,DEVICE_NAME,&fops);
104 
105   printk(KERN_INFO "UNGS : Driver registrado\n");
106   printk(KERN_INFO "I was assigned major number %d.To talk to the driver create a dev file with", Major);
107   printk(KERN_INFO "mknod -m 666 /dev/%s c %d 0\n", DEVICE_NAME, Major);
108   printk(KERN_INFO "Try to cat and echo to the device file.\n");
109   printk(KERN_INFO "Remove the device file and module when done.\n");
110 
111   return 0;
112 }
113 
114 void cleanup_module(void)
115 {
116   unregister_chrdev(Major,DEVICE_NAME);
117   printk(KERN_INFO "UNGS : Driver desregistrado\n");
118 }
```

4) Definir funciones *device_open* y *device_release*

Device_open es un contador que permite controlar las veces que se accede o se sale de un driver. Además, cada vez que abrimos el driver colocamos un puntero al inicio de un mensaje.

```
41 static int device_open(struct inode *inode, struct file *file)
42 {
43   Device_Open++;
44 
45   msg_Ptr = msg;
46 
47   printk(KERN_ALERT "se abrió el dirver.\n");
48 
49  return 0;
50 }
51 static int device_release(struct inode *inode, struct file *file)
52 {
53  Device_Open--;
54 
55  printk(KERN_ALERT "se cerro el driver.\n");
56  return 0;
57 }
```

5) Escribir y leer el char device

Antes de poder leer o escribir se crea una carpeta con el comando
Mknod –m 666 /dev/nombreDeDriver c majorNumber minorNumber

Mknod: nos permite crear la carpeta(dispositivo) para comunicarnos con el driver
-m 666: asignamos los permisos, en este caso es lectura y escritura para todos
/dev/nombreDeDriver: lugar donde se encuentra el driver
C: tipo de dispositivo. Existen los de tipo carácter(C) o de bloque(B)
MajorNumber: numero con el cual el file system diferencia los drivers
MinorNumber: es usado internamente por el driver, para saber que dispositivo se maneja

Para escribir, usamos el comando **echo** junto a una cadena y lo mandamos al dispositivo.
Para leer, usamos el comando **cat** y la ubicación del dispositivo

También, agregamos la vista del log del kernel donde se muestra que el driver fue escrito o leído respectivamente.
Salvo que en lugar de hacer uso del comando **dmesg,** con el comando **tail** listo las ultimas diez filas del archivo donde están los log del kernel.

```
alumno@alumno-virtualbox:~/sor2/TP0$ echo "Hola mundo" >> /dev/device_reverso
alumno@alumno-virtualbox:~/sor2/TP0$ tail -n 3 /var/log/syslog
Mar 26 14:13:57 alumno-virtualbox kernel: [11242.577092] se abrió el dirver.
Mar 26 14:13:57 alumno-virtualbox kernel: [11242.577147] se escribio.
Mar 26 14:13:57 alumno-virtualbox kernel: [11242.577156] se cerro el driver.
alumno@alumno-virtualbox:~/sor2/TP0$ cat /dev/device_reverso
Hola mundo
alumno@alumno-virtualbox:~/sor2/TP0$ tail -n 6 /var/log/syslog
Mar 26 14:13:57 alumno-virtualbox kernel: [11242.577092] se abrió el dirver.
Mar 26 14:13:57 alumno-virtualbox kernel: [11242.577147] se escribio.
Mar 26 14:13:57 alumno-virtualbox kernel: [11242.577156] se cerro el driver.
Mar 26 14:14:14 alumno-virtualbox kernel: [11259.739354] se abrió el dirver.
Mar 26 14:14:14 alumno-virtualbox kernel: [11259.739400] se leyó.
Mar 26 14:14:14 alumno-virtualbox kernel: [11259.740098] se cerro el driver.
```

6) Escribir y leer el char device, con el mensaje al revés

Utilizamos los mismos comandos para leer y escribir que en el ejemplo anterior.
Acá, la diferencia radica en el código C utilizado ya que se hace uso de una función auxiliar que permite ordenar las letras para que al momento de leerlas se muestren al revés.

Por último, se muestra como borrar el dispositivo, el driver y los archivos de manera segura.