TP 0 - Device Drivers

Licenciatura en Sistemas
Sistemas Operativos y Redes II
2024 - 1° Semestre

Alumno:
Giordana, Facundo Tahiel

Docentes:
Chuquimango Chilon, Luis Benjamin
Echabarri, Alan Pablo

Introducción

El siguiente trabajo se encuentra dividido en dos secciones. En la primera de ellas se realiza la carga de un módulo Hola Mundo provisto previamente, mientras que en la segunda se elabora un kernel module para un char device. Todo lo realizado durante este trabajo se encuentra almacenado en el siguiente repositorio:
https://github.com/TahielGiordana/tp0-device-drivers

Módulo Hola Mundo

El objetivo de esta sección es cargar el módulo provisto en el siguiente repositorio: https://bitbucket.org/sor2/tp0. Para esto, comenzaremos ubicando los archivos correspondientes y compilar el módulo mediante el uso del comando ***make***.

```
alumno@alumno-virtualbox:~/Descargas$ cd tp0-device-drivers/
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers$ ls
Makefile  miModulo.c  README.md
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers$ make
make -C /lib/modules/5.4.0-70-generic/build M=/home/alumno/Descargas/tp0-device-drivers modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-70-generic'
  CC [M]  /home/alumno/Descargas/tp0-device-drivers/miModulo.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/alumno/Descargas/tp0-device-drivers/miModulo.mod.o
  LD [M]  /home/alumno/Descargas/tp0-device-drivers/miModulo.ko
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-70-generic'
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers$
```

Notamos que se generó un nuevo archivo ***miModulo.ko***, el cual representa un kernel module, cuya información podemos consultar mediante el comando ***modinfo***.

```
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers$ modinfo miModulo.ko
filename:       /home/alumno/Descargas/tp0-device-drivers/miModulo.ko
description:    Un primer driver
author:         UNGS
license:        GPL
srcversion:     1CD920ACDE13DD71D14A866
depends:
retpoline:      Y
name:           miModulo
vermagic:       5.4.0-70-generic SMP mod_unload modversions
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers$
```

El siguiente paso es cargar el módulo en el kernel al ejecutar ***insmod ./miModulo.ko***. Los módulos cargados se listan en *proc/modules/,* por lo tanto podemos verificar que nuestro módulo se cargó de manera correcta, ya que lo vemos primero en la siguiente lista.

```
alumno@alumno-virtualbox:~$ cat /proc/modules
miModulo 16384 0 - Live 0x0000000000000000 (OE)
snd_intel8x0 45056 2 - Live 0x0000000000000000
snd_ac97_codec 131072 1 snd_intel8x0, Live 0x0000
ac97_bus 16384 1 snd_ac97_codec, Live 0x000000000
snd_pcm 106496 2 snd_intel8x0,snd_ac97_codec, Liv
snd_seq_midi 20480 0 - Live 0x0000000000000000
snd_seq_midi_event 16384 1 snd_seq_midi, Live 0x0
```

Por último, en la biografía se explica que, si se desea, se puede descargar el módulo mediante el uso de ***rmmod***.

Módulo Char Device

Durante la siguiente sección se abordará la elaboración de un kernel module para un char device, tomando como guía el siguiente material:
https://www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf

Comenzamos definiendo las funciones ***init_module*** y ***cleanup_module***.
La primera de ellas, ***init_module***, es la función que se ejecuta al cargar el módulo, en el caso del char device debe encargarse principalmente de registrar el dispositivo mediante la función ***register_chrdev(0, DEVICE_NAME, &fops).***

```c
/*
* This function is called when the module is loaded
*/
int init_module(void)
{
        Major = register_chrdev(0, DEVICE_NAME, &fops);

        if (Major < 0) {
            printk(KERN_ALERT "Registering char device failed with %d\n", Major);
            return Major;
        }

        printk(KERN_INFO "I was assigned major number %d. To talk to\n", Major);
        printk(KERN_INFO "the driver, create a dev file with\n");
        printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
        printk(KERN_INFO "Try various minor numbers. Try to cat and echo to\n");
        printk(KERN_INFO "the device file.\n");
        printk(KERN_INFO "Remove the device file and module when done.\n");

        return SUCCESS;
}
```

De manera contraria, ***cleanup_module*** se encarga de eliminar el registro del dispositivo, previamente a la descarga del módulo.

```c
/*
* This function is called when the module is unloaded
*/
void cleanup_module(void)
{
    /*
    * Unregister the device
    */
    int ret = unregister_chrdev(Major, DEVICE_NAME);
    if (ret < 0)
        printk(KERN_ALERT "Error in unregister_chrdev: %d\n", ret);
}
```

A continuación debemos definir las funciones ***device_open*** y ***device_release***, las cuáles en el caso del char device se encargan de incrementar/decrementar el contador que indica cuántos procesos están utilizando el módulo mediante las funciones ***try_module_get(THIS_MODULE)*** y ***module_put(THIS_MODULE)***.

```c
/*
* Called when a process tries to open the device file, like
* "cat /dev/mycharfile"
*/
static int device_open(struct inode *inode, struct file *file){
    static int counter = 0;

    if (Device_Open)
        return -EBUSY;

    Device_Open++;
    try_module_get(THIS_MODULE);

    return SUCCESS;
}

/*
* Called when a process closes the device file.
*/
static int device_release(struct inode *inode, struct file *file)
{
    Device_Open--;          /* We're now ready for our next caller */

    /*
    * Decrement the usage count, or else once you opened the file, you'll
    * never get get rid of the module.
    */
    module_put(THIS_MODULE);

    return 0;
}
```

Luego debemos hacer que nuestro char device imprima en el kernel cuando le escribimos, para esto definimos la función ***device_write***.

```c
/*
* Called when a process writes to dev file: echo "hi" > /dev/hello
*/
static ssize_t device_write(struct file *filp, const char *buff, size_t len, loff_t * off)
{
    int i;

    for (i=0; i < len && i < BUF_LEN; i++){
        get_user(msg[i], buff + i);
    }

    msg_Ptr = msg;
    printk(KERN_INFO "Msg received: %s\n", msg);
    return i;

}
```

Para verificar que funcione debemos repetir los pasos realizados en la primera sección para cargar el módulo. Una vez hecho esto podremos verificar que se reciben los mensajes enviados.

```
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$ echo "Hola CharDevice" > /dev/chardev
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$ dmesg | tail
[15954.946369] Remove the device file and module when done.
[16748.624254] I was assigned major number 240. To talk to
[16748.624255] the driver, create a dev file with
[16748.624255] 'mknod /dev/chardev c 240 0'.
[16748.624255] Try various minor numbers. Try to cat and echo to
[16748.624256] the device file.
[16748.624256] Remove the device file and module when done.
[16814.952271] Msg received: Hola CharDevice
```

El siguiente paso es hacer que el char device devuelva lo último que fue escrito, por lo tanto se debe definir la función ***device_read.***

```c
static ssize_t device_read(struct file *filp,   /* see include/linux/fs.h   */
                           char *buffer,        /* buffer to fill with data */
                           size_t length,       /* length of the buffer     */
                           loff_t * offset)
{
    /*
    * Number of bytes actually written to the buffer
    */
    int bytes_read = 0;

    /*
    * If we're at the end of the message, return 0 signifying end of file
    */
    if (*msg_Ptr == 0)
        return 0;

    /*
    * Actually put the data into the buffer
    */
    while (length && *msg_Ptr) {

        /*
        * The buffer is in the user data segment, not the kernel
        * segment so "*" assignment won't work.  We have to use
        * put_user which copies data from the kernel data segment to
        * the user data segment.
        */

        put_user(*(msg_Ptr++), buffer++);
        length--;
        bytes_read++;
    }
    /*
    * Most read functions return the number of bytes put into the buffer
    */

    return bytes_read;
}
```

De esta manera, podemos verificar utilizando *cat /dev/chardev* que se devuelve el mensaje enviado con el comando *echo* .

```
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$ echo "Hola CharDevice" > /dev/chardev
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$ dmesg | tail
              ou 0 times Hello world!

[17262.115478] I was assigned major number 240. To talk to
[17262.115478] the driver, create a dev file with
[17262.115479] 'mknod /dev/chardev c 240 0'.
[17262.115479] Try various minor numbers. Try to cat and echo to
[17262.115479] the device file.
[17262.115479] Remove the device file and module when done.
[17288.513933] Msg received: Hola CharDevice

alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$ cat /dev/chardev
Hola CharDevice
```

Por último se nos pide que nuestro mensaje sea devuelto con los caracteres al revés, es decir que nuestro mensaje original “Hola CharDevice” debería devolverse como “eciveDrahC aloH”. Para esto modificamos la función ***device_read*** indicando que recorra de manera inversa los caracteres, comenzando desde el último carácter registrado en la variable *msg_length* que representa la longitud del mensaje. A continuación podemos ver cómo resulta la nueva función.

```c
static ssize_t device_read(struct file *filp,   /* see include/linux/fs.h   */
                           char *buffer,        /* buffer to fill with data */
                           size_t length,       /* length of the buffer     */
                           loff_t * offset)
{
    /*
    * Number of bytes actually written to the buffer
    */
    int bytes_read = 0;
    int i = 0;
    /*
    * If we're at the end of the message,
    * return 0 signifying end of file
    */
    if (*msg_Ptr == 0)
        return 0;

    /*
    * Actually put the data into the buffer
    */
    for(i = msg_length - 1; i >= 0 && bytes_read < length; --i ){
        put_user(msg[i], buffer++);
        ++bytes_read;
        *(msg_Ptr)++;
    }
    *offset = bytes_read;
    /*
    * Most read functions return the number of bytes put into the buffer
    */
    return bytes_read;
}
```

Verificamos el resultado repitiendo los pasos previos, notamos que ahora el mensaje se devuelve de forma inversa.

```
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$ echo "Hola CharDevice" > /dev/chardev
alumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$ cat /dev/chardev

eciveDrahC aloHalumno@alumno-virtualbox:~/Descargas/tp0-device-drivers/charDriver$
```