Universidad Nacional de General Sarmiento

Informe

# TP 0 - Device Drivers

Módulo “Hola Mundo” y “Char Device”

**Alumno:**

Sia, Giovanni Antonio 42370427

**Profesor**:

Chuquimango, Benjamin
Echabarri, Alan Pablo Daniel

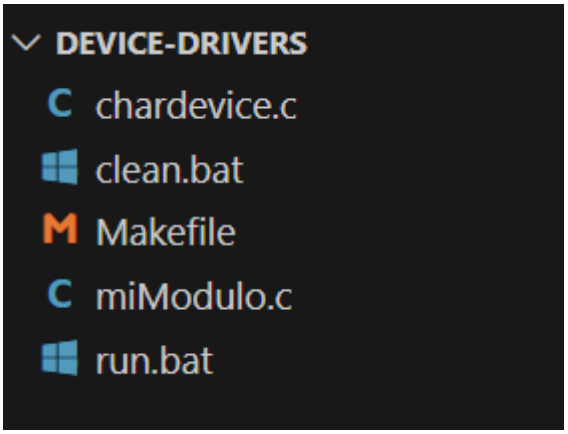
**Materia**:

Sistemas operativos y redes 2

# Introducción

En este informe se detalla la implementación de dos módulos de controladores de dispositivos en el kernel de Linux. Estos módulos son el “Hola Mundo”, el cual muestra un mensaje al cargar y descargar el módulo, y “Char Device”, que permite interactuar con el dispositivo a través de operaciones de lectura y escritura devolviendo así el texto ingresado por el usuario invertido. Se va describir el desarrollo de cada módulo, se analizarán los resultados y el cómo se obtuvieron.

# Archivos




Tenemos 5 archivos:

**chardevice.c:** Contiene el código fuente del módulo “Char Device”, el cual implementa un controlador de dispositivo de caracteres en el kernel de Linux.

**miModulo.c:** Contiene el código fuente del módulo “Hola Mundo”, que imprime mensajes en el registro del kernel al cargar y descargar el módulo.

**Makefile:** Archivo de configuración para compilar los módulos del kernel.

**run.bat:** Script para compilar y cargaFr los módulos en el kernel. Utilizado para automatizar el proceso de compilación, ahorrando así tiempo escribiendo cada comando en consola.

**clean.bat:** Script de limpieza para eliminar los archivos generados. Utilizado para automatizar el proceso de compilación, ahorrando así tiempo escribiendo cada comando en consola.

# Makefile

```
Makefile
1   obj-m += chardevice.o
2   obj-m += miModulo.o
3
4   all:
5       make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
6   clean:
7       make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
8
```

Como se puede ver compilamos los dos módulos, el Char Device y el “Hola Mundo”(el cual se llama “miModulo”).

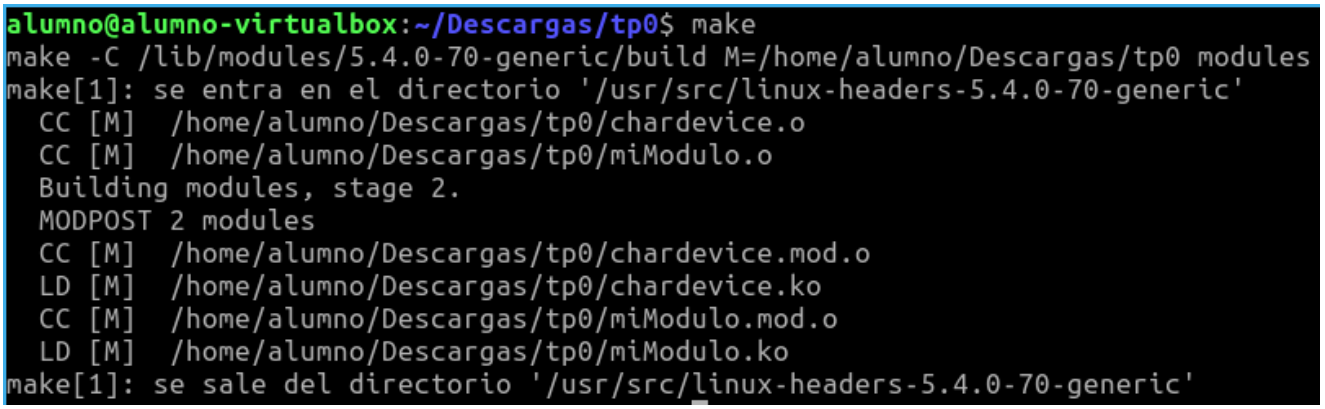
# Módulo “Hola Mundo”

Queremos verificar que funcione el módulo hola mundo obtenido del repositorio https://bitbucket.org/sor2/tp0 el cual nos proporciona el Makefile y el archivo miModulo.c, así que primero necesitamos compilarlo y cargarlo en el kernel de Linux. A continuación mostraré los comandos y como hacer para verificar que este módulo funciona.

Al código se le agregaron dos prints, el de “Hola Mundo” al cargar el módulo y el de “Chau Hola Mundo” al destruirlo.

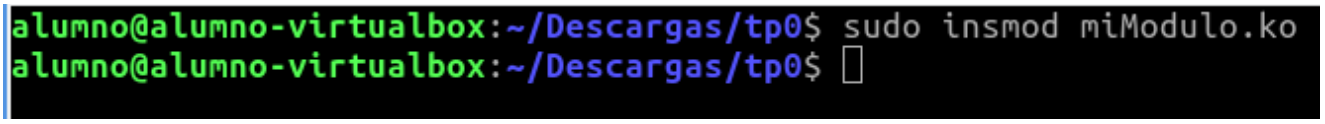
```
C miModulo.c > ...
1   #include <linux/module.h>
2   #include <linux/kernel.h>
3   int init_module(void)
4   { /* Constructor */
5     printk(KERN_INFO "UNGS : Driver registrado\n");
6     printk(KERN_INFO "Hola Mundo\n");
7     return 0;
8   }
9   void cleanup_module(void)
10  { /* Destructor */
11    printk(KERN_INFO "UNGS : Driver desregistrado\n");
12    printk(KERN_INFO "Chau Hola Mundo\n");
13  }
14  MODULE_LICENSE("GPL");
15  MODULE_AUTHOR("UNGS");
16  MODULE_DESCRIPTION("Un primer driver");
17
```

Primero en consola vamos a hacer “**make”** para compilar el módulo:

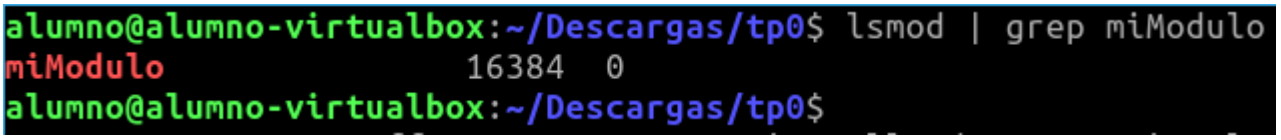



En esta imagen vemos que compilamos los dos módulos, pero nos interesa todo lo relacionado con “miModulo”.

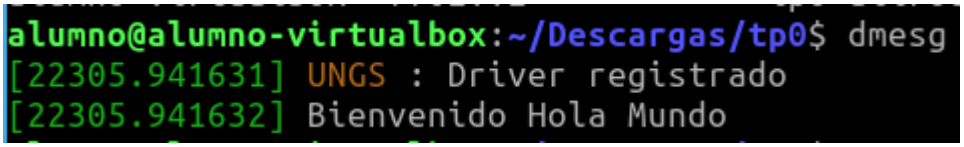
Una vez compilado correctamente vamos a cargar el módulo en el kernel usando “**insmod**”.



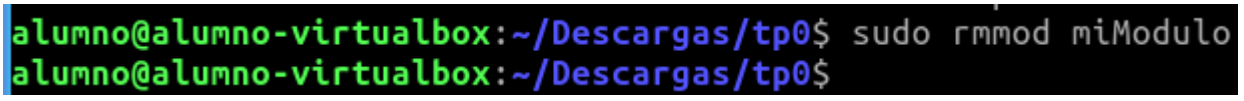

Ahora, para verificar que el módulo está cargado vamos a usar el comando **“lsmod”** el cual lista todos los módulos cargado actualmente en el kernel pero nosotros vamos a filtrar para obtener solo el módulo que queremos ver el cual es “miModulo”.




Una vez comprobado que el módulo “miModulo” está cargado en el kernel vamos a verificar los mensajes en el kernel, los cuales se encuentran en la función ***“init_module(void)”***. Estos mensajes se pueden ver mediante el comando **“dmesg”**.




Y por último vamos a descargar o destruir el módulo para ver los dos mensajes que se encuentra en la función ***“cleanup_module(void)”,*** para ello vamos a usar el comando **“rmmod”.**




Una vez descargado el módulo vamos a ver los mensajes en el kernel con **“dmesg”**.

```
alumno@alumno-virtualbox:~/Descargas/tp0$ dmesg
[22305.941631] UNGS : Driver registrado
[22305.941632] Bienvenido Hola Mundo
[22959.785130] UNGS : Driver desregistrado
[22959.785131] Chau Hola Mundo
alumno@alumno-virtualbox:~/Descargas/tp0$
```

De esta forma podemos verificar que el módulo ”miModulo” funciona.

# Módulo “Char Device”

Primero vamos a todas las funciones del código **“chardevice.c”** y al final mostraremos cómo funciona.

## Función init_module

Se ejecuta esta función cuando el módulo se carga en el kernel y realiza las siguiente acciones:

- Registra el dispositivo de caracteres usando ***“register_chrdev”***.
- Crea la clase del dispositivo utilizando ***“class_create”***. Se utiliza para crear una clase de dispositivo en el sistema, es como un contenedor para dispositivos relacionados.
- Crea el dispositivo en “/dev” utilizando ***“device_create”***. Se utiliza para crear un dispositivo asociado a una clase específica en el sistema.
- Al usar las funciones ***“class_create”*** y ***“device_create”***, no es necesario crear el dispositivo manualmente utilizando en consola **“mknod /dev/chardev [número mayor] [número menor]”**.

```
27  // Función que se ejecuta al cargar el módulo
28  int init_module(void) {
29      // Se registra el dispositivo de caracteres
30      major = register_chrdev(0, DEVICE_NAME, &chardev_fops);
31      if (major < 0) {
32          pr_alert("Falló el registro del dispositivo de caracteres con el número mayor %d\n", major);
33          return major;
34      }
35      pr_info("Dispositivo registrado con el número mayor %d.\n", major);
36
37      // Se crea la clase del dispositivo
38      cls = class_create(THIS_MODULE, DEVICE_NAME);
39      // Se crea el dispositivo en /dev
40      device_create(cls, NULL, MKDEV(major, 0), NULL, DEVICE_NAME);
41
42      pr_info("Dispositivo creado en /dev/%s\n", DEVICE_NAME);
43
44      return 0;
45  }
```

## Función cleanup_module

Se ejecuta esta función cuando el módulo se descarga del kernel y realiza las siguiente acciones:

- Destruye el dispositivo utilizando ***"device_destroy"***.
- Destruye la clase del dispositivo utilizando ***"class_destroy"***.
- Desregistra el dispositivo de caracteres utilizando ***"unregister_chrdev"***.

```
47  // Función que se ejecuta al descargar el módulo
48  void cleanup_module(void) {
49      // Se destruye el dispositivo
50      device_destroy(cls, MKDEV(major, 0));
51      // Se destruye la clase del dispositivo
52      class_destroy(cls);
53      // Se desregistra el dispositivo de caracteres
54      unregister_chrdev(major, DEVICE_NAME);
55
56      pr_info("Dispositivo desregistrado\n");
57  }
```

## Función device_open

Se ejecuta al abrir el dispositivo y realiza las siguiente acciones:

- Incrementa el contador de uso del módulo usando ***"try_module_get"***. Esto es para que el módulo no se destruya mientras haya procesos que estén utilizando el dispositivo.
- Retorna 0 indicando que el dispositivo se abrió correctamente y está listo para usarse.

```
59  // Función que se ejecuta al abrir el dispositivo
60  static int device_open(struct inode *inode, struct file *file) {
61      // Incrementa el contador de uso del módulo
62      try_module_get(THIS_MODULE);
63      return 0;
64  }
```

## Función device_release

Se ejecuta al cerrar el dispositivo y realiza las siguiente acciones:

- Decrementa el contador de uso del módulo usando ***"device_release"***.Esto le indica al kernel que el módulo ya no esta en uso y puede ser destruido si es necesario.
- Retorna 0 indicando que el dispositivo se cerró correctamente y los recursos fueron liberados.

```c
// Función que se ejecuta al cerrar el dispositivo
static int device_release(struct inode *inode, struct file *file) {
    // Decrementa el contador de uso del módulo
    module_put(THIS_MODULE);
    return 0;
}
```

## Función device_read

Se ejecuta al leer desde el dispositivo utilizando en consola **“cat /dev/chardev”**.

- Copia el texto almacenado en “msg” (el mensaje escrito previamente) al espacio de usuario, pero invierte el orden de los caracteres y lo transfiere al buffer del espacio de usuario utilizando ***“copy_to_user”***.

```c
// Función que se ejecuta al leer desde el dispositivo
static ssize_t device_read(struct file *filp, char __user *buffer, size_t length, loff_t *offset) {
    int bytes_read = 0;
    int msg_len = strlen(msg);

    // Si se leyó hasta el final del mensaje, se retorna 0
    if (*offset >= msg_len)
        return 0;

    // Calcula la cantidad de bytes a leer
    bytes_read = min(length, (size_t)(msg_len - *offset));

    // Copia el mensaje al espacio de usuario
    if (copy_to_user(buffer, msg + (msg_len - *offset - bytes_read), bytes_read))
        return -EFAULT;

    *offset += bytes_read;

    return bytes_read;
}
```

## Función device_write

Se ejecuta cuando se escribe en el dispositivo utilizando en consola **“echo “hola” > /dev/chardev”**. Toma los datos ingresados desde el espacio de usuario y los almacena en el buffer del módulo del kernel.

- Copia los datos desde el buffer del espacio de usuario al buffer del módulo del kernel usando **“copy_from_user”**.
- Invierte el texto ingresado por el usuario en “**msg”.**

```c
// Función que se ejecuta al escribir en el dispositivo
static ssize_t device_write(struct file *filp, const char __user *buff, size_t len, loff_t *off) {
    int bytes_written = 0;
    int start = 0;
    int end = len - 1;
    char temp;
    if (len > BUF_LEN) {
        pr_alert("El texto es demasiado largo.\n");
        return -EINVAL;
    }
    // Copia el texto desde el espacio de usuario al buffer del módulo
    if (copy_from_user(msg, buff, len)) {
        pr_alert("Error al copiar desde el espacio de usuario.\n");
        return -EFAULT;
    }
    pr_info("Texto recibido: %s\n", msg);
    // Evito invertir el carácter de salto de línea
    if (msg[end] == '\n') {
        end--;
    }
    // Invierte el texto
    while (start < end) {
        temp = msg[start];
        msg[start] = msg[end];
        msg[end] = temp;
        start++;
        end--;
    }
    bytes_written = len;
    pr_info("Texto invertido: %s\n", msg);
    return bytes_written;
}
```

## Ejecutar “Char Device”

Para verificar que funciona vamos primero a compilar el módulo. Ejecutamos en consola el comando **“make”** para compilar el módulo.

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

Solo nos fijamos en los archivos que tienen **“chardevice.”.**

Una vez compilado el módulo vamos a cargarlo en el sistema utilizando el comando **“insmod”**.

```
alumno@alumno-virtualbox:~/Descargas/tp0$ sudo insmod chardevice.ko
alumno@alumno-virtualbox:~/Descargas/tp0$
```

Verificamos que el módulo esté cargado mediante “lsmod” y vamos a usar un filtro para simplificar la búsqueda.

```
alumno@alumno-virtualbox:~/Descargas/tp0$ lsmod | grep chardevice
chardevice             16384  0
alumno@alumno-virtualbox:~/Descargas/tp0$
```

Ya comprobado que el módulo “chardevice” está cargado en el kernel vamos a verificar los mensajes en el kernel. Estos mensajes se pueden ver mediante el comando **“dmesg”**.

```
alumno@alumno-virtualbox:~/Descargas/tp0$ dmesg
[27955.374547] Dispositivo registrado con el número mayor 240.
[27955.377311] Dispositivo creado en /dev/chardev
alumno@alumno-virtualbox:~/Descargas/tp0$
```

Cómo creamos directamente el enlace mediante código entonces no tenemos que usar **“mknod /dev/chardev 240 0”.**

Le otorgamos permisos a “/dev/chardev” mediante **“chmod”.**

```
alumno@alumno-virtualbox:~/Descargas/tp0$ sudo chmod 666 /dev/chardev
```

Y vamos directamente a interactuar con el dispositivo. Vamos a escribir en el dispositivo utilizando **“echo”** y leer desde el dispositivo utilizando **“cat”**.

```
alumno@alumno-virtualbox:~/Descargas/tp0$ sudo echo "hola" > /dev/chardev
alumno@alumno-virtualbox:~/Descargas/tp0$ cat /dev/chardev
aloh
alumno@alumno-virtualbox:~/Descargas/tp0$
```

Finalmente vemos en la consola del kernel los datos mediante **“dmesg”.**

```
alumno@alumno-virtualbox:~/Descargas/tp0$ dmesg
[28618.706543] Dispositivo registrado con el número mayor 240.
[28618.709406] Dispositivo creado en /dev/chardev
[28716.393410] Texto recibido: hola

[28716.393411] Texto invertido: aloh
```

Finalmente para simplificar el ingresar todos los comandos para limpiar el módulo vamos a ejecutar el script **“clean.bat”** pero como super usuario usando **“sudo su”** primero**.**

```
root@alumno-virtualbox:/home/alumno/Descargas/tp0# . clean.bat
make -C /lib/modules/5.4.0-70-generic/build M=/home/alumno/Descargas/tp0 clean
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-70-generic'
  CLEAN   /home/alumno/Descargas/tp0/Module.symvers
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-70-generic'
[29272.702430] Dispositivo registrado con el número mayor 240.
[29272.705131] Dispositivo creado en /dev/chardev
[29272.709427] UNGS : Driver registrado
[29272.709428] Bienvenido Hola Mundo
[29272.712395] Texto recibido: hola

[29272.712397] Texto invertido: aloh

[29278.708153] Dispositivo desregistrado
[29278.710293] UNGS : Driver desregistrado
[29278.710293] Chau Hola Mundo
root@alumno-virtualbox:/home/alumno/Descargas/tp0#
```

**Adjunto github:** [https://github.com/GiovanniSia/Device-Drivers](https://github.com/GiovanniSia/Device-Drivers)