

Trabajo Práctico

Device Drivers

Sistemas Operativos y Redes II

Alumno:

Sanchez Rodriguez, Camila (41024628/2018)

Docentes:

Chuquimango Chilon Luis Benjamin

Echabarri Alan Pablo Daniel

Aula:

7230

Periodo:

Primer semestre 2024

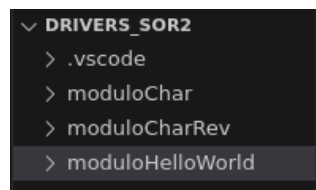
Introducción

El presente trabajo tiene como objetivo principal profundizar en el funcionamiento de los módulos en el kernel de Linux. Los módulos desempeñan un papel muy importante al actuar como intermediarios vitales entre el hardware y el software, permitiendo una interacción fluida y eficiente entre los usuarios y las máquinas.

Para la realización de este trabajo, se proporcionó material teórico¹ para comprender la estructura y el funcionamiento de los componentes esenciales de un módulo. Además, se brindó el esqueleto² de código para la implementación de un primer módulo *“Hola Mundo”* y la posterior construcción de un módulo *“Char Device”*.

Mediante esta combinación teórica-práctica se buscará dar una visión integral de los drivers tratando de dejar sentando las bases de este tema dentro del ámbito de los sistemas operativos.

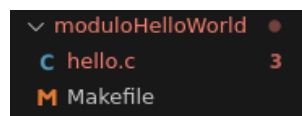
Con antelación se aclara se que decidió el siguiente formato para ordenar los archivos de manera tal que se pueda mostrar y ejecutar los puntos solicitados en el trabajo práctico. Cada carpeta representa un módulo y en cada una de ellas se presentará el código C del driver y su respectivo archivo Makefile.




Por último, se utilizará la plataforma GitHub para crear el repositorio³ que contendrá los archivos pertinentes del trabajo práctico, junto con el correspondiente informe.


Módulo Hola Mundo

Utilizando el código provisto por los docentes, se realizaron los cambios necesarios para que nos devuelva un *“Hola Mundo”*. El código se encuentra en la carpeta *“moduloHelloWorld”* donde se puede encontrar un archivo en código C *hello.c* y un archivo Makefile que nos facilitará la compilación y el enlace de nuestro archivo C.



En el archivo *hello.c* se crea un driver el cual simplemente presenta un mensaje *“Hello World!”* en el buffer del kernel al ser cargado en el mismo. A su vez, el módulo muestra un mensaje *“Goodbye World!”* al ser retirado de los kernel modules. Ambos mensajes están generados por las funciones de *init_module()* y *cleanup_module()*.

¹  linux kernel module (2023).pdf

²  tp0.pdf

³ https://github.com/lc-sanchez/Drivers_SOR2

```
C hello.c 3 x
moduloHelloWorld > C hello.c > ...
1  #include <linux/module.h>
2  #include <linux/kernel.h>
3
4  int init_module(void){
5      /*Constructor*/
6      pr_info("Hello World! \n");
7      /*A non 0 return means init_module failed; module can't be loaded*/
8      return 0;
9  }
10
11 void cleanup_module(void){
12     /*Destructor*/
13     pr_info("Goodbye World! \n");
14 }
15
16 MODULE_LICENSE("GPL");
17 MODULE_AUTHOR("Camila Sanchez");
18 MODULE_DESCRIPTION("Modulo Hola Mundo");
```

En cuanto al archivo *Makefile* tenemos el siguiente código:

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

Para la carga del módulo, nos ubicamos en la carpeta *moduloHelloWorld* y ejecutamos el comando *make*.

```
● alumno@alumno-virtualbox:~/TPS/Drivers_SOR2$ ls
  moduloChar  moduloCharRev  moduloHelloWorld
● alumno@alumno-virtualbox:~/TPS/Drivers_SOR2$ cd moduloHelloWorld
● alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ make
make -C /lib/modules/5.4.0-174-generic/build M=/home/alumno/TPS/Drivers_SOR2/moduloHelloWorld modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-174-generic'
  CC [M]  /home/alumno/TPS/Drivers_SOR2/moduloHelloWorld/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/alumno/TPS/Drivers_SOR2/moduloHelloWorld/hello.mod.o
  LD [M]  /home/alumno/TPS/Drivers_SOR2/moduloHelloWorld/hello.ko
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-174-generic'
○ alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$
```

Como se puede observar el archivo compila y crea una serie de archivos, entre ellos el archivo *hello.ko*. Ahora bien, es necesario instalar nuestro módulo *helloWorld* y para esto utilizamos el comando ***sudo insmod ./hello.ko***. Posteriormente, para verificar que el módulo se haya instalado correctamente ejecutamos el comando ***sudo lsmod*** antes y después de ejecutar el comando *insmod*, el cual nos permite observar una lista de todos los módulos del kernel cargando en el sistema hasta ese momento.

```

• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ sudo lsmod
Module                Size  Used by
ufs                    81920  0
qnx4                   16384  0
hfsplus                110592  0
hfs                    61440  0
minix                  36864  0
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ sudo insmod ./hello.ko
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ sudo lsmod
Module                Size  Used by
hello                 16384  0
ufs                    81920  0
qnx4                   16384  0
hfsplus                110592  0

```

Como se puede observar nuestro módulo ha sido cargado correctamente. Ahora se procederá a eliminar nuestro módulo mediante el comando **sudo rmmod ./hello.ko** para luego verificar mediante el comando **sudo dmesg** que nuestro módulo haya mostrado los mensajes correctamente.

```

• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ sudo rmmod ./hello.ko
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ sudo dmesg
[ 3852.500258] Hello World!
[ 3860.488016] Goodbye World!
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ █

```

Como se puede observar nuestro módulo muestra correctamente los mensajes en el buffer del kernel. Sin embargo, no hay que olvidar ejecutar el comando **make clean** para poder limpiar nuestro directorio de los archivos que hayan sido generados durante el proceso de compilación.

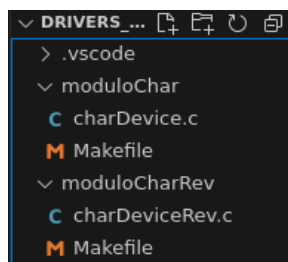
```

• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloHelloWorld$ make clean
make -C /lib/modules/5.4.0-174-generic/build M=/home/alumno/TPS/Drivers_SOR2/moduloHelloWorld clean
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-174-generic'
CLEAN /home/alumno/TPS/Drivers_SOR2/moduloHelloWorld/Module.symvers
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-174-generic'

```

Módulo Char Device

En esta parte del trabajo se pretende la implementación de un Char Device. Debido a que son múltiples puntos solicitados, se decidió dividir en dos carpetas cada una con su código C y su Makefile correspondiente. Ambos archivos C contienen un código similar siendo su única diferencia el hecho en “CharDevice” imprime en el buffer del kernel lo que se haya escrito sin realizar cambio alguno mientras que en “CharDeviceRev” imprimirá lo ingresado de manera reversa.



En ambos casos, como se explicó en el caso del módulo *Hello*, las funciones *init_module()* y *cleanup_module()* se encargaran de cargar y retirar el módulo de la lista de módulos del kernel. En este caso, en el *init_module()* se necesita asignar un *major number* al módulo durante la inicialización, dicho número sirve como identificador para el driver, es decir, indica qué tipo de device driver está asociado

Sistemas Operativos y Redes II - Trabajo Práctico 0

con el módulo del kernel. Se utilizó en la función **register_chrdev** el número 0 como parámetro para garantizar una asignación dinámica y evitar la posibilidad de que se establezca uno ya ocupado.

```
41
42  /*When the module is loaded*/
43  int init_module(void)
44  {
45      major = register_chrdev(0,DEVICE_NAME,&fops);
46
47      if(major <0){
48          pr_info("Registro de char device ha fallado con %d\n",major);
49          return major;
50      }
51      else {
52          pr_info("Se ha asignado el major number %d\n",major);
53          pr_info("Cree un dev file con mknod /dev/%s c %d 0\n",DEVICE_NAME,major);
54      }
55      return SUCCESS;
56  }
```

Respecto a la función *cleanup_module* se utiliza la función **unregister_chardev** en el cual se le ingresa como parámetro el major number utilizado para liberarlo.

```
57
58  /*When the module is unloaded*/
59  void cleanup_module(void)
60  {
61      //se saca del registro al device
62      unregister_chrdev(major,DEVICE_NAME);
63      pr_info("Se ha quitado el modulo %s. \n",DEVICE_NAME);
64  }
65
```

A continuación se definieron las funciones de *device_open()* y *device_release()* las cuales se ejecutarán al intentar abrir y cerrar un archivo del device respectivamente. En ambas funciones se utiliza un contador para chequear si el device está abierto y evitar múltiples accesos. Además, se utiliza la función **try_module_get** para incrementar el contador de referencias del módulo cada vez que un proceso acceda a él. Dicho contador tiene como propósito preservar nuestro módulo de que sea removido mientras esté en uso. Asimismo, el método **module_put()** en *device_release()* se utiliza para decrementar el contador de referencias del módulo, lo que permite que el kernel sepa que el módulo ya no es necesario y puede ser descargado de la memoria. Una vez que el contador de referencias alcanza el 0 puede ser removido sin problemas.

```
68  /*When a process tries to open the device file like "cat /dev/mycharfile" */
69  static int device_open(struct inode *inode, struct file *file)
70  {
71      if(Device_open){
72          return -EBUSY;
73      }
74      Device_open++;
75      try_module_get(THIS_MODULE);
76      return SUCCESS;
77  }
78  /*When a process closes the device file*/
79  static int device_release(struct inode *inode, struct file *file)
80  {
81      Device_open--; //NO estamos listos para la siguiente llamada
82      module_put(THIS_MODULE);
83      return SUCCESS;
84  }
```

En el siguiente punto, se pretendió que nuestro `charDevice` muestre en el kernel aquello que le escribiéramos. Por lo tanto, en el método `device_write()` se utilizó la función **`copy_from_user`** la cual se utiliza para copiar datos desde el espacio de memoria del usuario (user space) a un espacio de memoria del kernel (kernel space). Luego, se utilizó mediante un **`pr_info()`** para mostrar lo escrito en nuestro kernel space.

```
static ssize_t device_write(struct file *filp, const char __user *buff, size_t len, loff_t *off)
{
    procfs_buffer_size = len;
    if(len > PROCFS_MAX_SIZE){
        procfs_buffer_size = PROCFS_MAX_SIZE;
    }

    /*Limpiamos el buffer del kernel para la nueva escritura*/
    memset(procfs_buffer,0,sizeof procfs_buffer);

    /*se copia datos del user space al kernel space*/
    if (copy_from_user(procfs_buffer,buff,procfs_buffer_size)){
        return -EFAULT;
    }
    /*Agregamos elemento nulo*/
    procfs_buffer[procfs_buffer_size]='\0';
    /*Actualizamos desplazamiento*/
    *off+=procfs_buffer_size;

    pr_info("Se ha escrito correctamente: %s \n", procfs_buffer);
    pr_info("Escritura terminada.\n");

    return procfs_buffer_size;
}
```

Por otro lado, se pide posteriormente que se pueda visualizar lo último escrito por lo tanto también se implementó el método **`device_read()`** el cual se utiliza la función **`copy_to_user`** para transferir el mensaje desde el kernel space hacia el user space para después mostrarlo por pantalla mediante el comando **`cat`**.

```
static ssize_t device_read(struct file *filp, char *buffer, size_t length, loff_t *offset)
{
    if(*offset >= procfs_buffer_size){
        pr_info("Lectura terminada. \n");
        return SUCCESS; //termino lectura
    }

    /*se copia el contenido del kernel al user space */
    if (copy_to_user(buffer,procfs_buffer,procfs_buffer_size)){
        return -EFAULT; //si hay error al copiar los datos
    }

    /*Actualizamos la posicion del offset*/
    *offset += procfs_buffer_size;

    pr_info("Se lee lo siguiente: %s \n",buffer);
    return procfs_buffer_size;
}
```

La carga del módulo `charDevice` se realiza ejecutando los mismos comandos que al cargar el módulo `hello`, por lo que se mostrará a continuación todos los comandos ejecutados.

```

• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2$ cd moduloChar
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ make
make -C /lib/modules/5.4.0-174-generic/build M=/home/alumno/TPS/Drivers_SOR2/moduloChar modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-174-generic'
CC [M] /home/alumno/TPS/Drivers_SOR2/moduloChar/charDevice.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/alumno/TPS/Drivers_SOR2/moduloChar/charDevice.mod.o
LD [M] /home/alumno/TPS/Drivers_SOR2/moduloChar/charDevice.ko
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-174-generic'

• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ sudo insmod ./charDevice.ko
[sudo] contraseña para alumno:
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ sudo lsmod
Module                Size  Used by
charDevice             16384  0

```

En este punto se ejecuta el comando **sudo mknod /dev/charDevice c 240 0**, dicho comando crea un archivo de device el cual representa nuestro charDevice en el sistema indicando que su mayor number es 240 y el menor number es 0.

```

[ 7983.393976] Cree un dev file con mknod /dev/charDevice c 240 0
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ sudo mknod /dev/charDevice c 240 0

```

Ahora tenemos que agregar permisos de escritura por lo que ejecutamos el comando **sudo chmod 777 /dev/charDevice**. Seguidamente, efectuamos los comandos **echo "hola" >> /dev/charDevice**, el cual escribe "hola" en el charDevice, y el comando **cat /dev/charDevice** que se encargará de mostrar el contenido de nuestro device en la terminal.

```

• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ sudo chmod 777 /dev/charDevice
[sudo] contraseña para alumno:
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ echo "hola" >> /dev/charDevice
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ cat /dev/charDevice
hola
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$

```

Para observar que también lo imprime en el kernel space, realizamos el comando **sudo dmesg** y podremos observar que si imprime lo que hayamos escrito en el kernel.

```

[15430.033601] Se ha escrito correctamente: hola
[15430.033601] Escritura terminada.

```

Finalmente, removemos el módulo con los comando **sudo rmmod ./charDevice.ko**, **sudo rm /dev/charDevice** para borrar el archivo del device y **make clean** para limpiar el directorio.

```

• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ sudo rm /dev/charDevice
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ make clean
make -C /lib/modules/5.4.0-174-generic/build M=/home/alumno/TPS/Drivers_SOR2/moduloChar clean
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-174-generic'
CLEAN /home/alumno/TPS/Drivers_SOR2/moduloChar/Module.symvers
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-174-generic'
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloChar$ sudo lsmod
Module                Size  Used by
ufs                   81920  0
qnx4                   16384  0
hfsplus               110592  0

```



```
[15779.161081] Se ha quitado el modulo charDevice.
```

Para concluir, se decidió hacer el device *moduloCharRev* para la funcionalidad de devolver el mensaje al revés. Este módulo comparte el mismo código que el módulo *CharDevice*, la única diferencia es que en el método *device_read()* se le agrega un método auxiliar llamado *reverso* que se encarga de dar vuelta el mensaje que reciba.

```
/*Method to reverse a msg*/
void reverso(char *buffer){
    int i=0;
    int j=procfs_buffer_size-2;
    char letra;

    if(buffer == NULL){
        return; //No hay msg que procesar
    }

    //Ahora vamos intercambiando las letras
    while(i<j){
        letra = buffer[i];
        buffer[i]=buffer[j];
        buffer[j]=letra;
        i++;
        j--;
    }
}
```

```
static ssize_t device_read(struct file *filp, char *buffer, size_t length, loff_t *offset)
{
    if(*offset >= procfs_buffer_size){
        pr_info("Lectura terminada. \n");
        return SUCCESS; //termino lectura
    }

    /*se copia el contenido del kernel al user space*/
    if (copy_to_user(buffer,procfs_buffer,procfs_buffer_size)){
        return -EFAULT; //si hay error al copiar los datos
    }

    //Damos vuelta el mensaje
    reverso(buffer);

    //Actualizamos la posicion del offset
    *offset += procfs_buffer_size;

    pr_info("Se lee lo siguiente: %s \n",buffer);
    return procfs_buffer_size;
}
```

Para demostrar el funcionamiento del device, cargamos el módulo, creamos el archivo especial del device, se brinda los permisos necesarios y ejecutamos los comandos **echo** y **cat** para poder visualizar el resultado.

```
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2$ cd moduloCharRev
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ make
make -C /lib/modules/5.4.0-174-generic/build M=/home/alumno/TPS/Drivers_SOR2/moduloCharRev modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-174-generic'
CC [M] /home/alumno/TPS/Drivers_SOR2/moduloCharRev/charDeviceRev.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/alumno/TPS/Drivers_SOR2/moduloCharRev/charDeviceRev.mod.o
LD [M] /home/alumno/TPS/Drivers_SOR2/moduloCharRev/charDeviceRev.ko
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-174-generic'
```


Sistemas Operativos y Redes II - Trabajo Práctico 0

```
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ sudo insmod ./charDeviceRev.ko
[sudo] contraseña para alumno:
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ sudo dmesg
[16983.936623] Se ha asignado el mayor number 240
[16983.936624] Cree un dev file con mknod /dev/charDeviceRev c 240 0
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ sudo mknod /dev/charDeviceRev c 240 0
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ sudo chmod 777 /dev/charDeviceRev
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ echo "hola mundo" >> /dev/charDeviceRev
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ cat /dev/charDeviceRev
odnum aloh
```

Como se puede observar, funciona correctamente proporcionando el mensaje al revés. Finalmente, retiramos el módulo y damos por terminado la demostración.

```
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ sudo rmmod ./charDeviceRev.ko
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ sudo rm /dev/charDeviceRev
• alumno@alumno-virtualbox:~/TPS/Drivers_SOR2/moduloCharRev$ make clean
make -C /lib/modules/5.4.0-174-generic/build M=/home/alumno/TPS/Drivers_SOR2/moduloCharRev clean
make[1]: se entra en el directorio '/usr/src/linux-headers-5.4.0-174-generic'
CLEAN /home/alumno/TPS/Drivers_SOR2/moduloCharRev/Module.symvers
make[1]: se sale del directorio '/usr/src/linux-headers-5.4.0-174-generic'
```

```
[17046.710030] Se ha escrito correctamente: hola mundo
[17046.710030] Escritura terminada.
[17056.234722] Se lee lo siguiente: odnum aloh
[17056.234733] Lectura terminada.
[17149.588382] Se ha quitado el modulo charDeviceRev.
```