Universidad Nacional de Córdoba

Facultad de Ciencias Exactas Físicas y Naturales

Trabajo Práctico #5

“Drivers”

´

Asignatura: Sistemas de computación

APELLIDO Y NOMBRE	CARRERA	MATRICULA
Corvalán, Abel Nicolás	Ingeniería Electrónica	41.220.050
Soria, Federico Isaia	Ingeniería en computación	40.574.892

2024

Índice

Marco teórico..... 3
Drivers..... 3
CDD y CDF..... 3
Pruebas de drivers..... 4
Drv1.c..... 4
Drv2.c..... 6
Drv3.c..... 9
Drv4.c..... 11
Clipboard.ko..... 13
Implementación en Raspberry PI emulado en Qemu..... 15
Conclusiones..... 16
Bibliografía..... 17

Marco teórico

Drivers

Se trata de un software que permite al sistema operativo interactuar con un periférico, creando una abstracción del hardware y proporcionando una interfaz -posiblemente estandarizada- para utilizarlo. Se puede graficar como un manual de instrucciones que indica cómo controlar y comunicarse con un dispositivo en particular.

CDD y CDF

Los CDD (Character Device Drivers, o Controladores de Dispositivos de Carácter) son un tipo de controlador de dispositivo en el sistema operativo Linux que manejan dispositivos que transmiten datos de manera secuencial, es decir, carácter por carácter.
A diferencia de los controladores de bloques, que manejan dispositivos que acceden a los datos en bloques (como discos duros), los CDD leen y escriben datos carácter por carácter.
Estos drivers interactúan directamente con el kernel de Linux y pueden responder a interrupciones de hardware, administrar buffers de entrada y salida, y proporcionar interfaces de ioctl (entrada/salida controlada) para operaciones específicas del dispositivo.

Drivers y buses

LLos device controllers se conectan al bus del sistema

Pruebas de drivers

Drv1.c

Como vimos en el trabajo práctico anterior, cualquier driver de Linux consta de un constructor y un destructor.
Se llama al constructor de un módulo cada vez que insmod logra cargar el módulo en el núcleo y al destructor del módulo cada vez que rmmod logra descargar el módulo del núcleo.
Estas funciones se implementan con las macros module_init() y module_exit() incluidas en el encabezado de module.h.

Primero buscamos los drivers en el primer enlace proporcionado y para comenzar se trabajará con el **drv1.c, éste es un módulo simple ya que sólo imprime un mensaje al registro del kernel al ser cargado y otro mensaje al ser descargado sin otra operación adicional.**

drv1.c

```c
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>

static int __init drv1_init(void) /* Constructor */
{
    printk(KERN_INFO "SdeC: drv1 Registrado exitosamente..!!\n");

    return 0;
}

static void __exit drv1_exit(void) /* Destructor */
{
    printk(KERN_INFO "SdeC: drv1 dice Adios mundo cruel..!!\n");
}

module_init(drv1_init);
module_exit(drv1_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Anil Kumar Pugalia <email@sarika-pugs.com>");
MODULE_DESCRIPTION("Nuestro primer driver de SdeC");
```

Se procede a compilar con **“make all”** y se verifica la información del módulo mediante **“modinfo”**,de este modo se pueden ver atributos como la descripción, el autor o la versión.

```
fede@fede-VirtualBox:~/Escritorio/Drv1$ make all
make -C /lib/modules/5.15.0-106-generic/build M=/home/fede/Escritorio/Drv1 modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.15.0-106-generic'
  CC [M]  /home/fede/Escritorio/Drv1/drv1.o
  MODPOST /home/fede/Escritorio/Drv1/Module.symvers
  CC [M]  /home/fede/Escritorio/Drv1/drv1.mod.o
  LD [M]  /home/fede/Escritorio/Drv1/drv1.ko
  BTF [M] /home/fede/Escritorio/Drv1/drv1.ko
Skipping BTF generation for /home/fede/Escritorio/Drv1/drv1.ko due to unavailability of vmlinux
make[1]: se sale del directorio '/usr/src/linux-headers-5.15.0-106-generic'
```

Se inserta el módulo en el kernel mediante **“sudo insmod”**, con **“dmeg”** se verifica su correcta instalación y la impresión del mensaje.

```
fede@fede-VirtualBox:~/Escritorio/Drv1$ modinfo drv1.ko
filename:       /home/fede/Escritorio/Drv1/drv1.ko
description:    Nuestro primer driver de SdeC
author:         Anil Kumar Pugalia <email@sarika-pugs.com>
license:        GPL
srcversion:     2CCA21E672E4755AF8B453B
depends:
retpoline:      Y
name:           drv1
vermagic:       5.15.0-106-generic SMP mod_unload modversions
```

```
fede@fede-VirtualBox:~/Escritorio/Drv1$ sudo insmod drv1.ko
fede@fede-VirtualBox:~/Escritorio/Drv1$ sudo dmesg
[    0.000000] Linux version 5.15.0-106-generic (buildd@lcy02-amd64-017) (gcc (Ubuntu 11.4.0-1ub
untu1~22.04) 11.4.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #116-Ubuntu SMP Wed Apr 17 09:17:56
UTC 2024 (Ubuntu 5.15.0-106.116-generic 5.15.149)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-106-generic root=UUID=2679ede2-30be
-4d5d-bc26-26a7af89fe7e ro quiet splash
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
```

```
[    5.898656] 17:44:41.918843 main     vbglR3GuestCtrlDetectPeekGetCancelSupport: Supported (#1
)
[   10.101218] ISO 9660 Extensions: Microsoft Joliet Level 3
[   10.111170] ISO 9660 Extensions: RRIP_1991A
[ 1302.671968] SdeC: drv1 Registrado exitosamente..!!
```

Como sucede con cualquier módulo, se remueve luego con **rmmod** por lo que se imprime el mensaje de la función **printk** del destructor **drv1_exit()**.

```
[ 1302.671968] SdeC: drv1 Registrado exitosamente..!!
[ 1418.942470] SdeC: drv1 dice Adios mundo cruel..!!
```

Drv2.c

Del mismo modo que antes, compilamos el drv2 y se revisa su información.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
fede@fede-VirtualBox:~/Escritorio/Drv2$ make all
make -C /lib/modules/5.15.0-106-generic/build M=/home/fede/Escritorio/Drv2 modul
es
make[1]: se entra en el directorio '/usr/src/linux-headers-5.15.0-106-generic'
  CC [M]  /home/fede/Escritorio/Drv2/drv2.o
  MODPOST /home/fede/Escritorio/Drv2/Module.symvers
  CC [M]  /home/fede/Escritorio/Drv2/drv2.mod.o
  LD [M]  /home/fede/Escritorio/Drv2/drv2.ko
make[1]: se sale del directorio '/usr/src/linux-headers-5.15.0-106-generic'
```

```
fede@fede-VirtualBox:~/Escritorio/Drv2$ modinfo drv2.ko
filename:       /home/fede/Escritorio/Drv2/drv2.ko
description:    Nuestro segundo driver de SdeC
author:         Cátedra Sistemas de Computación
license:        GPL
srcversion:     89F44D7B1B9C483AF7A0465
depends:
retpoline:      Y
name:           drv2
vermagic:       5.15.0-106-generic SMP mod_unload modversions
```

Se inserta el módulo y se verifica que se imprimió correctamente el mensaje del constructor:

```
fede@fede-VirtualBox:~/Escritorio/Drv2$ sudo insmod drv2.ko
[sudo] contraseña para fede:
fede@fede-VirtualBox:~/Escritorio/Drv2$ sudo dmesg
[sudo] contraseña para fede:
[    0.000000] Linux version 5.15.0-106-generic (buildd@lcy02-amd64-017) (gcc (U
buntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #116
-Ubuntu SMP Wed Apr 17 09:17:56 UTC 2024 (Ubuntu 5.15.0-106.116-generic 5.15.149
)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.15.0-106-generic root=UU
ID=2679ede2-30be-4d5d-bc26-26a7af89fe7e ro quiet splash
```

```
[ 3673.296599] SdeC_drv2 Registrado exitosamente..!!
[ 3673.296602] <Major, Minor>: <237, 0>
```

A diferencia del Drv1, se observan 2 números en el registro, esto es debido a que este módulo registra dinámicamente un rango de números de dispositivo de carácter al cargarse y se libera ese rango al descargarse. **¿Pero qué son estos números?**
Cada archivo de dispositivo tiene asociado un par de números conocidos como major y minor.

- Major Number: Identifica el controlador de dispositivo que maneja las operaciones para ese dispositivo, o dicho de otra manera, **varios dispositivos del mismo tipo (controlados por el mismo controlador) compartirán el mismo número mayor.**
- Minor Number: **Este valor identifica el dispositivo específico gestionado por el controlador. por lo que diferentes dispositivos del mismo tipo tendrán diferentes números menores.**

En la función de inicialización (drv2_init), se reserva un número mayor y tres números menores para el dispositivo, imprimiendo un mensaje de éxito y los números asignados en el registro del kernel. En la función de limpieza (drv2_exit), se libera el rango de números de dispositivo y se imprime un mensaje de despedida.
El numero **major** es 237 y el **minor**, 0. Esto se puede revisar también mediante **cat /proc/devices** donde se observa que el driver se cargó en el major 237.

```
226 drm
237 SdeC_Driver2
238 aux
```

Al realizarse la asignación dinámica del número mayor (major number), no se puede anticipar cuál será el número asignado hasta que el módulo sea cargado por lo que dentro de la carpeta /dev **no hay archivos creados para el driver.**

```
fede@fede-VirtualBox:~/Escritorio/Drv2$ ls /dev/
autofs           loop5          tty0   tty4       ttyS11   vboxguest
block            loop6          tty1   tty40      ttyS12   vboxuser
bsg              loop7          tty10  tty41      ttyS13   vcs
btrfs-control    loop-control   tty11  tty42      ttyS14   vcs1
bus              mapper         tty12  tty43      ttyS15   vcs2
cdrom            mcelog         tty13  tty44      ttyS16   vcs3
char             mem            tty14  tty45      ttyS17   vcs4
console          mqueue         tty15  tty46      ttyS18   vcs5
core             net            tty16  tty47      ttyS19   vcs6
cpu              null           tty17  tty48      ttyS2    vcs7
cpu_dma_latency  nvram          tty18  tty49      ttyS20   vcsa
cuse             port           tty19  tty5       ttyS21   vcsa1
disk             ppp            tty2   tty50      ttyS22   vcsa2
dma_heap         psaux          tty20  tty51      ttyS23   vcsa3
dri              ptmx           tty21  tty52      ttyS24   vcsa4
ecryptfs         pts            tty22  tty53      ttyS25   vcsa5
fb0              random         tty23  tty54      ttyS26   vcsa6
fd               rfkill         tty24  tty55      ttyS27   vcsa7
full             rtc            tty25  tty56      ttyS28   vcsu
fuse             rtc0           tty26  tty57      ttyS29   vcsu1
hidraw0          sda            tty27  tty58      ttyS3    vcsu2
hpet             sda1           tty28  tty59      ttyS30   vcsu3
hugepages        sda2           tty29  tty6       ttyS31   vcsu4
hwrng            sda3           tty3   tty60      ttyS4    vcsu5
i2c-0            sg0            tty30  tty61      ttyS5    vcsu6
initctl          sg1            tty31  tty62      ttyS6    vcsu7
input            shm            tty32  tty63      ttyS7    vfio
kmsg             snapshot       tty33  tty7       ttyS8    vga_arbiter
log              snd            tty34  tty8       ttyS9    vhci
loop0            sr0            tty35  tty9       udmabuf  vhost-net
loop1            stderr         tty36  ttyprintk  uhid     vhost-vsock
loop2            stdin          tty37  ttyS0      uinput   zero
loop3            stdout         tty38  ttyS1      urandom  zfs
loop4            tty            tty39  ttyS10     userio
```

```c
static int __init drv2_init(void) /* Constructor */
{
    int ret;

    printk(KERN_INFO "SdeC_drv2 Registrado exitosamente..!!");


    if ((ret = alloc_chrdev_region(&first, 0, 3, "SdeC_Driver2")) < 0)
    {
        return ret;
    }
    printk(KERN_INFO "<Major, Minor>: <%d, %d>\n", MAJOR(first), MINOR(first));
    return 0;
}
```

La implementación de esta asignación dinámica llama a la función **alloc_chrdev_region** para asignar dinámicamente un rango de números de dispositivo de carácter. Los parámetros de alloc_chrdev_region son:

- **&first:** Puntero a una variable dev_t donde se almacenará el primer número de dispositivo asignado.
- **0**: Número minor inicial en el rango.
- **3**: Cantidad de números minor en el rango (en este caso, se solicitan 3 números minors).
- **"SdeC_Driver2":** Nombre del dispositivo (se usa para crear los archivos de dispositivo en /dev).

A continuación se crean entonces los ficheros del character device driver con mknod, asignando distintos MINORS al MAYOR ya determinado.

```
fede@fede-VirtualBox:~/Escritorio/Drv2$ sudo mknod /dev/drv2_0 c 237 0
[sudo] contraseña para fede:
fede@fede-VirtualBox:~/Escritorio/Drv2$ sudo mknod /dev/drv2_1 c 237 1
fede@fede-VirtualBox:~/Escritorio/Drv2$ sudo mknod /dev/drv2_2 c 237 2
```

Observamos que se crearon correctamente los device files mediante **grep**:

```
fede@fede-VirtualBox:~/Escritorio/Drv2$ ls /dev/ | grep drv2
drv2_0
drv2_1
drv2_2
```

Pero sin embargo, si se intenta abrir uno de estos archivos aparece un mensaje de error ya que en este código no tienen implementadas las funciones de open() y close().

```
fede@fede-VirtualBox:~/Escritorio/Drv2$ cat /dev/drv2_0
cat: /dev/drv2_0: No existe el dispositivo o la dirección
```

Drv3.c

Compilamos sin problemas el driver 3, se obtiene su información y se agrega al kernel.

```
fede@fede-VirtualBox:~$ cd Escritorio/Drv3
fede@fede-VirtualBox:~/Escritorio/Drv3$ make all
make -C /lib/modules/5.15.0-106-generic/build M=/home/fede/Escritorio/Drv3 modul
es
make[1]: se entra en el directorio '/usr/src/linux-headers-5.15.0-106-generic'
  CC [M]  /home/fede/Escritorio/Drv3/drv3.o
  MODPOST /home/fede/Escritorio/Drv3/Module.symvers
  CC [M]  /home/fede/Escritorio/Drv3/drv3.mod.o
  LD [M]  /home/fede/Escritorio/Drv3/drv3.ko
make[1]: se sale del directorio '/usr/src/linux-headers-5.15.0-106-generic'
```

```
fede@fede-VirtualBox:~/Escritorio/Drv3$ modinfo drv3.ko
filename:       /home/fede/Escritorio/Drv3/drv3.ko
description:    Nuestro tercer driver de SdeC
author:         Cátedra Sistemas de Computación
license:        GPL
srcversion:     E61026D79A36AB5AF383D74
depends:
retpoline:      Y
name:           drv3
vermagic:       5.15.0-106-generic SMP mod_unload modversions
```

```
fede@fede-VirtualBox:~/Escritorio/Drv3$ sudo insmod drv3.ko
fede@fede-VirtualBox:~/Escritorio/Drv3$ sudo dmesg
```

Se verifica su correcto agregado por el mensaje en su module_init().

```
[ 8731.286377] SdeC_drv3 Registrado exitosamente..!!
```

Hasta ahora todo se comporta como los drivers anteriores pero tiene una gran diferencia y es que este código **define operaciones básicas de dispositivo como abrir, cerrar, leer y escribir**, imprimiendo mensajes en el registro del kernel cuando se realizan estas operaciones. Todas estas definidas dentro de **file_operations** como se muestra a continuación y en el constructor drv3_init se crea el puntero a un objeto de la clase device particular mediante:**cl = class_create(THIS_MODULE, “chardrv”)**

```
static struct file_operations pugs_fops =
{
    // Dentro de file_operations defino las funciones que voy a implementar..!!

    .owner = THIS_MODULE,
    .open = my_open,
    .release = my_close,
    .read = my_read,
    .write = my_write
};
```

- **`.owner = THIS_MODULE`**: Es un puntero al módulo del kernel que contiene estas operaciones. Mantiene una referencia al módulo para evitar que se descargue mientras alguna de estas operaciones está en curso.:
- **`open = my_open:`** Esta función se llama cuando un proceso abre el archivo de dispositivo. En este caso, imprime un mensaje en el registro del kernel indicando que el archivo de dispositivo ha sido abierto.

- **`release = my_close:`** Esta función se llama cuando un proceso cierra el archivo de dispositivo. Este también imprime un mensaje en el registro del kernel indicando que el archivo de dispositivo ha sido cerrado.
- **`read = my_read:`** Esta función se llama cuando un proceso intenta leer desde el archivo de dispositivo. Es responsable de transferir datos desde el kernel al espacio de usuario. También imprime un mensaje indicando que se ha realizado una operación de lectura. Aunque no transfiere datos ya que la implementación real debería copiar datos desde el dispositivo al búfer del usuario.
- **`write = my_write:`** Esta función se llama cuando un proceso intenta escribir en el archivo de dispositivo. Es responsable de transferir datos desde el espacio de usuario al kernel. En el código, imprime un mensaje indicando que se ha realizado una operación de escritura y devuelve el número de bytes escritos (que es `len`, el tamaño de los datos proporcionados por el usuario).

En las carpeta /sys/class/chardrv/SdeC_drv3 el kernel coloca información del CDD y se tiene una carpeta /sys/devices/virtual/chardrv/ donde están los dispositivos relacionados.

```
fede@fede-VirtualBox:~/Escritorio/Drv3$ find /sys -name Sde*
find: ‘/sys/kernel/tracing’: Permiso denegado
find: ‘/sys/kernel/debug’: Permiso denegado
/sys/class/chardrv/SdeC_drv3
/sys/devices/virtual/chardrv/SdeC_drv3
find: ‘/sys/fs/pstore’: Permiso denegado
find: ‘/sys/fs/bpf’: Permiso denegado
```

El character device driver creado “ SdeC_drv3” se encuentra en la carpeta /dev.

```
fede@fede-VirtualBox:~/Escritorio/Drv3$ ls -al /dev/SdeC_drv3
crw------- 1 root root 237, 0 may 28 17:17 /dev/SdeC_drv3
```

Con sudo cat /dev/SdeC_drv3 intentamos abrir el archivo, y con dmesg vemos que se cargaron las operaciones open(), read() y close():

```
fede@fede-VirtualBox:~/Escritorio/Drv3$ sudo cat /dev/SdeC_drv3
[sudo] contraseña para fede:
fede@fede-VirtualBox:~/Escritorio/Drv3$ sudo dmesg
[    0.000000] Linux version 5.15.0-107-generic (buildd@lcy02-amd64-012) (gcc (U
buntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #117
-Ubuntu SMP Fri Apr 26 12:26:49 UTC 2024 (Ubuntu 5.15.0-107.117-generic 5.15.149
)
```

```
[ 7568.355479] Driver3_SdeC: open()
[ 7568.355486] Driver3_SdeC: read()
[ 7568.355490] Driver3_SdeC: close()
```

Quitando el módulo con **“sudo rmmod drv3.ko”** , vemos que se imprime el mensaje del destructor:

```
[11529.869361] Driver3_SdeC: open()
[11529.869410] Driver3_SdeC: write()
[11529.869413] Driver3_SdeC: close()
[11576.212711] SdeC_drv3 dice Adios mundo cruel..!!
```

Drv4.c

Se define una variable global char c que se utiliza para almacenar un único carácter, esto no está presente en drv3. **A diferencia de drv3, en este caso si se realizan operaciones de escritura y lectura,** mediante las funciones **my_read** que lee el valor del carácter y lo copia al espacio de usuario utilizando copy_to_user. Si la lectura es exitosa, devuelve 1 y actualiza el offset; de lo contrario, devuelve un error.
Por otro lado, la función **my_write** toma el último byte del buffer del usuario y lo almacena en c utilizando copy_from_user. Si la escritura es exitosa, devuelve la longitud del búfer; de lo contrario, devuelve un error.

Algunas de funciones que se ejecutan dentro de la inicialización del módulo son:

int alloc_chrdev_region(dev_t * dev, unsigned baseminor, unsigned count, const char * name);
Esta se encarga de registrar un rango de números (menores) para dispositivos de caracteres.
Parámetros que la función requiere:

- **dev:** Guarda el número major en la variable first de la cual se pasa su dirección de memoria para ser seteada dentro del cuerpo de la función.
- **baseminor:** Guarda el primero del rango de valores menores. En nuestro caso 0
- **count:** La cantidad de números menores requeridas. Para nuestro caso solo 1
- **name:** Nombre del device driver. En nuestro caso SdeC_drv4

struct class* class_create (struct module* owner, const char* name);
Crea una estructura que sirve de argumento para otra función llamada device_create()
Recibe como argumentos:

- **owner:** Un puntero al módulo que posee esta estructura. En este caso es el mismo módulo que estamos trabajando.
- **name:** Un nombre representativo para el nombre de esta estructura.

int cdev_add (struct cdev* p, dev_t dev, unsigned count);
Agrega un dispositivo de caracteres al sistema.
Recibe como parámetros:

- **p:** Estructura retornada por el llamado a la función cdev_init .
- **dev:** Número de device obtenido de alloc_chrdev_region.
- **count:** Cantidad de números menores para este dispositivo. En nuestro caso es solo uno.

Compilamos con make como en los casos anteriores y con **modinfo** verificamos su información general..

```
fede@fede-VirtualBox:~/Escritorio/Drv4$ make
make -C /lib/modules/5.15.0-107-generic/build M=/home/fede/Escritorio/Drv4 modul
es
make[1]: se entra en el directorio '/usr/src/linux-headers-5.15.0-107-generic'
  CC [M]  /home/fede/Escritorio/Drv4/drv4.o
  MODPOST /home/fede/Escritorio/Drv4/Module.symvers
  CC [M]  /home/fede/Escritorio/Drv4/drv4.mod.o
  LD [M]  /home/fede/Escritorio/Drv4/drv4.ko
  BTF [M] /home/fede/Escritorio/Drv4/drv4.ko
Skipping BTF generation for /home/fede/Escritorio/Drv4/drv4.ko due to unavailabi
lity of vmlinux
make[1]: se sale del directorio '/usr/src/linux-headers-5.15.0-107-generic'
```

```
fede@fede-VirtualBox:~/Escritorio/Drv4$ modinfo drv4.ko
filename:       /home/fede/Escritorio/Drv4/drv4.ko
description:    Nuestro cuarto driver de SdeC
author:         Cátedra Sistemas de Computación
license:        GPL
srcversion:     8403FEA5EDDEB944500144E
depends:
retpoline:      Y
name:           drv4
vermagic:       5.15.0-107-generic SMP mod_unload modversions
```

Se agrega el módulo al kernel y se verifica su instalación

```
fede@fede-VirtualBox:~/Escritorio/Drv4$ sudo insmod drv4.ko
fede@fede-VirtualBox:~/Escritorio/Drv4$ sudo rmmod drv4.ko
fede@fede-VirtualBox:~/Escritorio/Drv4$ sudo dmesg
```

```
[ 3714.461498] SdeC_drv4: Registrado exitosamente..!!
[ 3752.300727] SdeC_drv4: dice Adios mundo cruel..!!
```

Por último, se escribe en el archivo del device driver mediante echo y se verifica con cat.

```
fede@fede-VirtualBox:~/Escritorio/Drv4$ sudo -i
[sudo] contraseña para fede:
root@fede-VirtualBox:~# echo -n "H" > /dev/SdeC_drv4
root@fede-VirtualBox:~# cat /devV/SdeC_drv4
```

```
root@fede-VirtualBox:~# cat /dev/SdeC_drv4
Hroot@fede-VirtualBox:~# exit
cerrar sesión
```

Clipboard.ko

Este módulo de kernel, a diferencia del drv4, está diseñado para proporcionar una funcionalidad de portapapeles accesible a través del sistema de archivos **/proc**. Este portapapeles permite a las aplicaciones de espacio de usuario leer y escribir datos en un buffer asignado dinámicamente en el espacio de kernel como se comprobará más adelante.

Cuando se carga el módulo, se reserva un espacio en memoria para el buffer del portapapeles utilizando **vmalloc** y se inicializa a ceros. Además, se crea una entrada en **/proc** llamada **clipboard** con permisos de lectura y escritura (**0666**). Las operaciones de lectura y escritura están definidas en las funciones **clipboard_read** y **clipboard_write**.

```c
int init_clipboard_module( void )
{
  int ret = 0;
  clipboard = (char *)vmalloc( BUFFER_LENGTH );

  if (!clipboard) {
    ret = -ENOMEM;
  } else {

    memset( clipboard, 0, BUFFER_LENGTH );
    proc_entry = proc_create( "clipboard", 0666, NULL, &proc_entry_fops);
    if (proc_entry == NULL) {
      ret = -ENOMEM;
      vfree(clipboard);
      printk(KERN_INFO "Clipboard: No puede crear entrada en /proc..!!\n");
    } else {
      printk(KERN_INFO "Clipboard: Modulo cargado..!!\n");
    }
  }
  return ret;
}
```

struct proc_dir_entry *proc_create (const char *name, umode_t mode, struct proc_dir_entry *parent, const struct file_operations *proc_fops)
Tiene el propósito de crear una entrada en el directorio proc . Recibe los parámetros:

- **name:** Nombre para la nueva entrada en proc . “Clipboard” en nuestro caso.
- **mode:** Permisos de acceso. “0666” en nuestro caso. Lo que se traduce en permisos de lectura y escritura para usuario , grupo y otro .
- **parent:** Nombre del directorio padre de la nueva entrada dentro de proc .
- **proc_fops:** Estructura que contiene punteros a función que se implementan en el driver. Es el reemplazo de la estructura deprecada para versiones más viejas del
- **kernel file_operations:** En nuestro caso definimos clipboard_read y clipboard_wr.

```c
static const struct proc_ops proc_entry_fops = {
        .proc_read = clipboard_read,
        .proc_write = clipboard_write,
};
```

Se compila y se carga el módulo, esto se verifica mediante **sudo dmesg.**

```
fede@fede-VirtualBox:~/Escritorio/Clipboard$ make all
make -C /lib/modules/5.15.0-107-generic/build M=/home/fede/Escritorio/Clipboard
modules
make[1]: se entra en el directorio '/usr/src/linux-headers-5.15.0-107-generic'
  CC [M]  /home/fede/Escritorio/Clipboard/clipboard.o
  MODPOST /home/fede/Escritorio/Clipboard/Module.symvers
  CC [M]  /home/fede/Escritorio/Clipboard/clipboard.mod.o
  LD [M]  /home/fede/Escritorio/Clipboard/clipboard.ko
  BTF [M] /home/fede/Escritorio/Clipboard/clipboard.ko
```

```
fede@fede-VirtualBox:~/Escritorio/Clipboard$ sudo insmod clipboard.ko
[sudo] contraseña para fede:
fede@fede-VirtualBox:~/Escritorio/Clipboard$ sudo dmesg | tail -n 2
[ 6043.170334] clocksource: Long readout interval, skipping watchdog check: cs_n
sec: 1068929770 wd_nsec: 1068929452
[ 6284.241111] Clipboard: Modulo cargado..!!
```

Se transfiere información desde el espacio de usuario al espacio del kernel creando una entrada en **/proc/clipboard** con **echo "mensaje" > /proc/clipboard.** Mediante **cat** se puede recuperar ese mensaje.

```
fede@fede-VirtualBox:~/Escritorio/Clipboard$ echo "Hola mundo" > /proc/clipboard
fede@fede-VirtualBox:~/Escritorio/Clipboard$ cat /proc/clipboard
Hola mundo
```

Cuando el módulo se descarga, se libera el espacio en memoria reservado para el buffer y se elimina la entrada en **`/proc`.**

```
fede@fede-VirtualBox:~/Escritorio/Clipboard$ sudo rmmod clipboard.ko
fede@fede-VirtualBox:~/Escritorio/Clipboard$ sudo dmesg | tail -n 2
[ 6331.833441] clocksource: Long readout interval, skipping watchdog check: cs_n
sec: 2157096339 wd_nsec: 2157095697
[ 6800.233983] Clipboard: Modulo descargado..!!
```

Implementación en Raspberry PI emulado en Qemu

Para esta etapa del trabajo práctico se necesita compilar un fork de Qemu con soporte de interrupciones GPIO. Para esto realizamos un fork del QEMU proporcionado por la cátedra. Se utilizó el comando **build** para crear el directorio y compilarlo.

```
fede@fede-VirtualBox:~/Escritorio$ git clone git@github.com:BlastNeos/qemu.git Q
EMU
Clonando en 'QEMU'...
remote: Enumerating objects: 517204, done.
remote: Total 517204 (delta 0), reused 0 (delta 0), pack-reused 517204
Recibiendo objetos: 100% (517204/517204), 302.02 MiB | 7.59 MiB/s, listo.
Resolviendo deltas: 100% (419031/419031), listo.
```

```
fede@fede-VirtualBox: ~/Escritorio/QEMU/build
 Archivo  Editar  Ver  Buscar  Terminal  Ayuda
fede@fede-VirtualBox:~$ cd Escritorio/QEMU
fede@fede-VirtualBox:~/Escritorio/QEMU$ mkdir build
fede@fede-VirtualBox:~/Escritorio/QEMU$ cd build
```

```
fede@fede-VirtualBox:~/Escritorio$ cd QEMU
fede@fede-VirtualBox:~/Escritorio/QEMU$ mkdir build
fede@fede-VirtualBox:~/Escritorio/QEMU$ cd build
```

Sin embargo, en la instalación tuvimos múltiples errores que se solucionaron con la instalación de las dependencias correspondientes.

Errores presentados:

```
fede@fede-VirtualBox:~/Escritorio/QEMU/build$ ../configure

ERROR: Cannot find Ninja
```

```
fede@fede-VirtualBox:~/Escritorio/QEMU/build$ ../configure

ERROR: glib-2.48 gthread-2.0 is required to compile QEMU
```

```
C++ linker for the host machine: c++ ld.bfd 2.38
Program cgcc found: NO
Library m found: YES
Library util found: YES
Run-time dependency appleframeworks found: NO (tried framework)
Found pkg-config: /usr/bin/pkg-config (0.29.2)
Run-time dependency pixman-1 found: NO (tried pkgconfig)

../meson.build:302:2: ERROR: Dependency "pixman-1" not found, tried pkgconfig

A full log can be found at /home/fede/Escritorio/QEMU/build/meson-logs/meson-log
.txt

ERROR: meson setup failed
```

Posterior a la solución de los errores, se procedió a instalar el paquete qemu-rpi-gpio a través de pip, el gestor de paquetes para Python. Este paquete proporciona soporte para los pines GPIO de la Raspberry Pi en QEMU, lo que permite emular la funcionalidad de los mismos dentro del entorno de QEMU.

```
fede@fede-VirtualBox:~$ pip install qemu-rpi-gpio
Defaulting to user installation because normal site-packages is not writeable
Collecting qemu-rpi-gpio
  Downloading qemu_rpi_gpio-0.4-py3-none-any.whl (16 kB)
Requirement already satisfied: pexpect in /usr/lib/python3/dist-packages (from q
emu-rpi-gpio) (4.8.0)
Installing collected packages: qemu-rpi-gpio
Successfully installed qemu-rpi-gpio-0.4
```

Se continúan instalando las dependencias necesarias.

```
fede@fede-VirtualBox:~$ sudo apt install python3-pexpect socat p7zip-full
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
python3-pexpect ya está en su versión más reciente (4.8.0-2ubuntu1).
```

Se descarga la **imagen de Raspbian**, ésta es un archivo que contiene el sistema operativo Raspbian, **diseñado para ejecutarse en dispositivos Raspberry Pi**. Lo clonamos desde el repositorio y lo instalamos.
Este archivo de imagen es lo que nos permitirá emular un entorno de Raspberry Pi en QEMU.

```
fede@fede-VirtualBox:~$ git clone https://github.com/javierajorge/qemu-rpi-gpio
Clonando en 'qemu-rpi-gpio'...
remote: Enumerating objects: 54, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 54 (delta 8), reused 6 (delta 4), pack-reused 39
Recibiendo objetos: 100% (54/54), 24.36 KiB | 489.00 KiB/s, listo.
Resolviendo deltas: 100% (20/20), listo.
fede@fede-VirtualBox:~$ cd qemu-rpi-gpio
fede@fede-VirtualBox:~/qemu-rpi-gpio$ ./qemu-pi-setup/setup.sh
Obj:1 https://packages.microsoft.com/repos/code stable InRelease
Obj:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Obj:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Obj:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Obj:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Ign:6 http://packages.linuxmint.com virginia InRelease
Obj:7 http://packages.linuxmint.com virginia Release
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
```

Establecemos una contraseña mediante echo, de modo que:

```
fede@fede-VirtualBox:~/qemu-rpi-gpio$ echo ContraseñaSegura | openssl passwd -6
-stdin
$6$V6gKg2QeamJDqxOs$F6K8cxL9uBh5HJE2JYjHAmg8mjot..CtZzUuhW36yGvuJ4HaVWEkg79lDgnl
Vg1WbBKJuFvRy8XqJQ5nw4IQQ.
```

```
fede@fede-VirtualBox:~/qemu-rpi-gpio$ python3 qemu-rpi-gpio
[ ] Virtual GPIO manager
[ ] Listening for connections
(gpio)>

(gpio)>
```

```
fede@fede-VirtualBox:~/qemu-rpi-gpio$ ./qemu-pi-setup/run.sh
+ QEMU=qemu-system-aarch64
+ cd /home/fede
+ ROOTFS=rootfs
+ TARGET=raspios_lite_armhf_latest.zip
+ SSHPORT=50022
+ ENABLEQTEST=true
+ QTESTSOCKET=/tmp/tmp-gpio.sock
```

```
fede@fede-VirtualBox:~/qemu-rpi-gpio/qemu-pi-setup$ ./setup.sh
[sudo] contraseña para fede:
Obj:1 https://packages.microsoft.com/repos/code stable InRelease
Obj:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:3 http://packages.linuxmint.com virginia InRelease
Obj:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Obj:5 http://packages.linuxmint.com virginia Release
Obj:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Obj:8 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
```

```
fede@fede-VirtualBox:~/qemu-rpi-gpio$ ./qemu-rpi-gpio
[ ] Virtual GPIO manager
[ ] Listening for connections
(gpio)>
```

```
fede@fede-VirtualBox:~/qemu-rpi-gpio/qemu-pi-setup$ ./run.sh
+ QEMU=qemu-system-aarch64
+ cd /home/fede
+ ROOTFS=rootfs
+ TARGET=raspios_lite_armhf_latest.zip
+ SSHPORT=50022
+ ENABLEQTEST=true
+ QTESTSOCKET=/tmp/tmp-gpio.sock
+ 7z l raspios_lite_armhf_latest.zip
+ awk /  raspios/{print $NF}
+ IMGNAME=raspios_lite_armhf_latest
```

Error de run.sh

```
qemu-system-aarch64: Invalid SD card size: 2.37 GiB
SD card size has to be a power of 2, e.g. 4 GiB.
You can resize disk images with 'qemu-img resize <imagefile> <new-size>'
(note that this will lose data if you make the image smaller than it currently i
s).
```

2do error de run.sh

```
+ qemu-system-aarch64 -display curses -display curses -serial stdio -M raspi3b -dtb root
fs/bcm2710-rpi-3-b-plus.dtb -kernel rootfs/kernel8.img -append  console=ttyAMA0,115200 e
arlyprintk loglevel=8 rw root=/dev/mmcblk0p2 rootwait rootfstype=ext4 -drive file=raspio
s_lite_armhf_latest,if=sd,format=raw,index=0 -device usb-net,netdev=net0 -netdev user,id
=net0,hostfwd=tcp:127.0.0.1:50022-:22 -qtest unix:/tmp/tmp-gpio.sock
qemu-system-aarch64: Failed to connect to '/tmp/tmp-gpio.sock': No such file or director
y
qemu-system-aarch64: Failed to initialize device for qtest: "unix:/tmp/tmp-gpio.sock"
```

```
fede@fede-VirtualBox:~$ ls -l /tmp-gpio.sock
srwxr-xr-x 1 root root 0 jun 14 15:09 /tmp-gpio.sock
```

```
abel@abel-asus:~$ qemu-system-aarch64 -monitor unix:/tmp/monitor.sock,server,nowait -nographic -machi
ne virt -cpu cortex-a57 -m 1024 -kernel ~/rootfs.orig/rootfs/kernel8.img -append "console=ttyAMA0" -n
etdev user,id=net0 -device virtio-net-device,netdev=net0
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x411fd070]
[    0.000000] Linux version 6.6.20+rpt-rpi-v8 (debian-kernel@lists.debian.org) (gcc-12 (Debian 12.2.
0-14) 12.2.0, GNU ld (GNU Binutils for Debian) 2.40) #1 SMP PREEMPT Debian 1:6.6.20-1+rpt1 (2024-03-0
7)
```

Bibliografía

- https://gitlab.com/sistemas-de-computacion-unc/device-drivers/
- https://stackoverflow.com/questions/71746914/linux-kernel-module-development-module-x86-modules-skipping-invalid-relocatio
- https://github.com/berdav/qemu
- https://archive.kernel.org/oldlinux/htmldocs/kernel-api/API-alloc-chrdev-region.html
- https://www.raspberrypi.com/news/raspberry-pi-bullseye-update-april-2022/