



UNC



Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales



Sistemas de Computación, 2023

Trabajo Práctico n° 5

Módulos del Kernel, parte 2

Grupo: Internautas

Integrantes:

- Careggio, Camila
- Casanueva, María Constanza
- Riba, Franco

Estudiantes de Ingeniería en Computación

Profesores: Ing. Jorge, Javier Alejandro y Solinas, Miguel

Índice

1. ¿Quien es Werner Almesberger?	3
2. Sistemas de Archivos FAT	4
3. Drivers	6
4. Character Device Drivers	7
5. Primer Character Device Driver	8
6. Análisis del source “drv2.c”	9
7. Análisis del source “drv3.c”	13
8. Trabajamos con el source “drv4.c”	16
9. Módulo “Clipboard” /proc	17

Repo: <https://github.com/francoriba/LinuxPiDriverDev>

1. ¿Quién es Werner Almesberger?

Buscamos el módulo vinculado al sistema de archivos (fs / fat) partiendo del directorio **/lib/modules/5.19.0-43-generic/kernel**

```
franco ~ cd /lib/modules/5.19.0-43-generic/kernel
franco /lib/modules/5.19.0-43-generic/kernel ls -l
total 56
drwxr-xr-x 3 root root 4096 jun  1 06:07 arch
drwxr-xr-x 2 root root 4096 jun  1 06:07 block
drwxr-xr-x 4 root root 4096 jun  1 06:07 crypto
drwxr-xr-x 112 root root 4096 jun  1 06:07 drivers
drwxr-xr-x 60 root root 4096 jun  1 06:07 fs
drwxr-xr-x 2 root root 4096 jun  1 06:07 kernel
drwxr-xr-x 9 root root 4096 jun  1 06:07 lib
drwxr-xr-x 2 root root 4096 jun  1 06:07 mm
drwxr-xr-x 60 root root 4096 jun  1 06:07 net
drwxr-xr-x 4 root root 4096 jun  1 06:07 samples
drwxr-xr-x 16 root root 4096 jun  1 06:07 sound
drwxr-xr-x 3 root root 4096 jun  1 06:07 ubuntu
drwxr-xr-x 2 root root 4096 jun  1 06:07 v4l2loopback
drwxr-xr-x 2 root root 4096 jun  1 06:07 zfs
franco /lib/modules/5.19.0-43-generic/kernel cd fs
/lib/modules/5.19.0-43-generic/kernel/fs
franco /lib/modules/5.19.0-43-generic/kernel/fs ls
9p      binfmt_misc.ko dlm      fscache jffs2 nfs_common omfs      reiserfs      ufs
adfs     btrfs      efs      fuse      jfs      nfsd      orangefs    romfs      vboxsf
affs     cachefiles eroofs   gfs2      ksmbd   nilfs2    overlayfs  shiftfs.ko  xfs
afs      ceph       exfat     hfs      lockd    nls      pstore      smbfs_common zonefs
autofs   cifs       f2fs     hfsplus   minix   ntfs      qnx4      sysv
befs     coda      fat      hpfs     netfs   ntfs3    qnx6      ubifs
btrfs   cramfs    freevxfs isofs    nfs     ocfs2    quota      udf
franco /lib/modules/5.19.0-43-generic/kernel/fs cd fat
/lib/modules/5.19.0-43-generic/kernel/fs/fat
franco /lib/modules/5.19.0-43-generic/kernel/fs/fat ls -l
total 32
-rw-r--r-- 1 root root 30305 may 19 11:45 msdos.ko
franco /lib/modules/5.19.0-43-generic/kernel/fs/fat
```

Luego usamos el comando **modinfo** con el módulo **msdos.ko**, y vemos que entre la información del módulo figura Werner Almesberger como autor.

```
franco /lib/modules/5.19.0-43-generic/kernel/fs/fat modinfo msdos.ko
filename:      /lib/modules/5.19.0-43-generic/kernel/fs/fat/msdos.ko
description:   MS-DOS filesystem support
author:        Werner Almesberger
license:      GPL
alias:        fs-msdos
srcversion:   E4EA6BAE3A40F24D67166E3
depends:
retpoline:    Y
intree:       Y
name:         msdos
vermagic:    5.19.0-43-generic SMP preempt mod_unload modversions
sig_id:       PKCS#7
signer:       Build time autogenerated kernel key
sig_key:      5F:3A:35:36:E6:C7:55:2F:41:0E:45:47:59:BD:05:31:24:7A:4D:29
sig_hashalgo: sha512
signature:    A8:64:0C:D8:71:07:7E:32:AD:95:C5:8F:5D:4E:51:E3:4E:A9:A3:11:
              02:58:80:2E:AC:0C:83:D2:EE:07:02:EE:D1:BA:08:85:3C:A1:BE:7E:
```

Lo stalkeamos en GitHub y descubrimos que está viviendo en Argentina.

The screenshot shows Werner Almesberger's GitHub profile. It features a circular profile picture of a man with dark hair. Below the picture, his name "Werner Almesberger" and GitHub handle "wpwvak" are displayed. A "Follow" button is present. A red box highlights the location information: "Buenos Aires, Argentina" with a small map icon and an email link "werner@almesberger.net". The "Popular repositories" section lists five repositories: "zc42-pic", "1104xe-pic", "idbg", "solvespace", and "ben-wpan-linux", all marked as "Public".

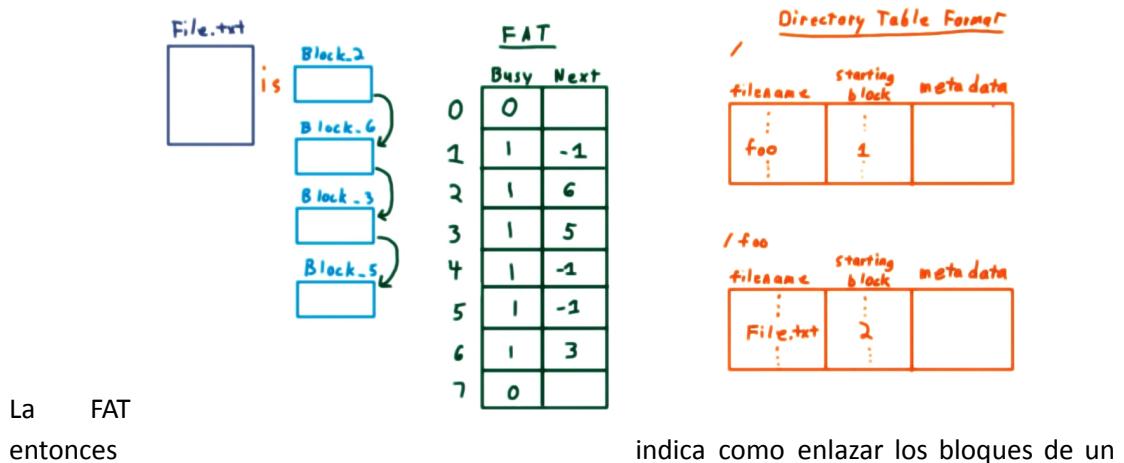
2. Sistemas de Archivos FAT

A continuación investigamos a grandes rasgos como funciona el sistema FAT, para complementar a lo comentado durante la clase.

El sistema de archivos **FAT** (File Allocation Table) se encuentra dividido en bloques y presenta un índice que indica donde se encuentra cada uno de los componentes de un archivo en particular.

Cada archivo se representa como una linked list de bloques, pero a diferencia de en una linked list tradicional, donde cada bloque/nodo contiene la referencia al siguiente, en el sistema FAT las referencias o enlaces al siguiente bloque se encuentran almacenadas en la “File Allocation Table” que es indexada mediante el número de bloque. De esta forma dado el numero de bloque inicial de un archivo se puede encontrar el siguiente bloque mediante un acceso constante a la FAT.

- Un valor especial en la tabla indica que un bloque es el ultimo en la cadena de bloques de un archivo.
- La FAT tambien contiene un bit “busy” que indica si un bloque se encuentra libre o no.



archivo, pero no indica donde comenzar la cadena de bloques, para ello hay tablas de directorio. Las tablas de directorio son tratadas como archivos y usan un “Formato de tabla de directorio”. Las entradas en la tabla tienen todos el mismo ancho y contienen la información esencial como:

- Nombre del archivo
- Bloque inicial
- Metadatos asociados al archivo como permisos

El directorio de archivos raíz tiene una dirección fija en el disco, por lo que siempre se sabe donde comenzar.

Existen sistemas de archivos más complejos que presentan más robustez a cuesta de pérdida de performance, como EXT3, NTFS (de código cerrado).

```
franco ~ cd /dev
franco /dev ls
autofs      i2c-5      loop30      nvme0n1p3  tty16  tty47      ttyS19      vcs2
block       i2c-6      loop31      nvme0n1p4  tty17  tty48      ttyS2       vcs3
bsg        i2c-7      loop32      nvram       port    tty18  tty49      ttyS20      vcs4
btrfs-control  iio:device0  loop33      port     tty19  tty5      ttyS21      vcs5
bus        initctl   loop34      ppp       tty2   tty50      ttyS22      vcs6
char       input     loop35      psaux     tty20  tty51      ttyS23      vcsa
console    kfd       loop36      ptmx      tty21  tty52      ttyS24      vcsal
core        kmsg    loop37      pts       tty22  tty53      ttyS25      vcsa2
cpu        kvm       loop38      random    tty23  tty54      ttyS26      vcsa3
cpu_dma_latency log     loop39      rfkill   tty24  tty55      ttyS27      vcsa4
cuse       loop0     loop4       rtc      tty25  tty56      ttyS28      vcsa5
disk       loop1     loop40      rtc0     tty26  tty57      ttyS29      vcsa6
dma_heap   loop10    loop41      sda      tty27  tty58      ttyS3       vcsu
dri        loop11    loop42      sda1     tty28  tty59      ttyS30      vcsul
drm_dp_aux0 loop12    loop43      sda2     tty29  tty6      ttyS31      vcsu2
cryptfs   loop13    loop44      sda3     tty3   tty60      ttyS4       vcsu3
fb0       loop14    loop5       sda4     tty30  tty61      ttyS5       vcsu4
fd        loop15    loop6       sda5     tty31  tty62      ttyS6       vcsu5
full      loop16    loop7       sg0      tty32  tty63      ttyS7       vcsu6
fuse       loop17    loop8       Shm     tty33  tty7      ttyS8       vfio
gpiochip0 loop18    loop9       snapshot  tty34  tty8      ttyS9       vga_arbiter
hidraw0   loop19    loop-control snd      tty35  tty9      udmabuf     vhci
hidraw1   loop2     mapper     stderr   tty36  ttyprintk  uhid       vhost-net
hidraw2   loop20    mcelog     stdin    tty37  tty50      uinput     vhost-vsock
hidraw3   loop21    media0     stdout   tty38  tty51      urandom    video0
hidraw4   loop22    mem       tty      tty39  tty510     usb       video1
hpet      loop23    mqueue    tty0     tty4   tty511     userio    zero
hugepages loop24    net       tty1     tty40  tty512     v4l       zfs
hw RNG    loop25    ng0n1     tty10    tty41  tty513     vboxdrv
i2c-0     loop26    null      tty11    tty42  tty514     vboxdrv
i2c-1     loop27    nvme0     tty12    tty43  tty515     vboxnetctrl
i2c-2     loop28    nvme0n1   tty13    tty44  tty516     vboxusb
i2c-3     loop29    nvme0n1p1  tty14    tty45  tty517     vcs
i2c-4     loop3     nvme0n1p2  tty15    tty46  tty518     vcs1
```

3. Drivers

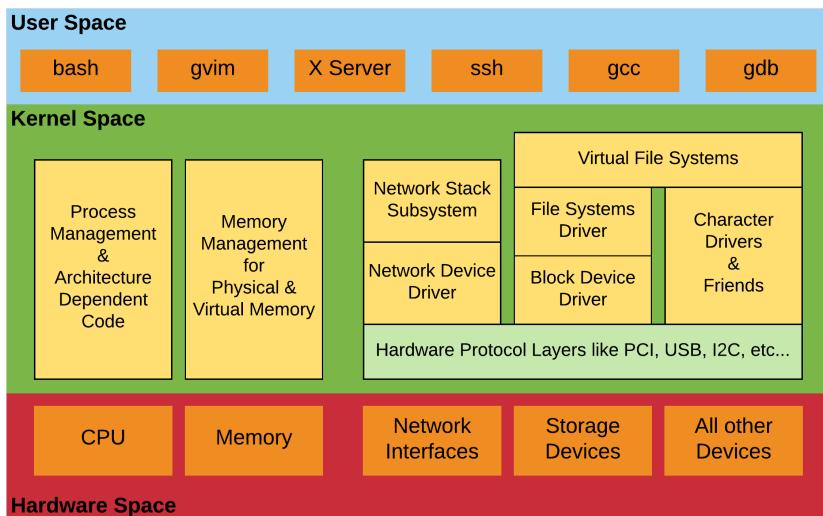
DEVICE DRIVER	DEVICE CONTROLLER
A computer program that operates or controls a particular type of device that is attached to a computer	A part of the computer system that makes sense of the signals going to and coming from the CPU
Software	Hardware
Works as a translator between the hardware device and the application or the operating system that uses it	Converts a serial bit stream to block of bytes and perform error correction as required

Específicamente en linux, podemos clasificar los drivers en tres tipos:

- Drivers de Networking (paquetes)
- Drivers de Storage (bloques)
- Drivers de Character (bytes)

En el Kernel de Linux podemos encontrar áreas enfocadas a:

- Gestión de procesos
- Gestión de memoria
- Stack de redes y
 - Soportado por los drivers de dispositivos de red
 - Soportados por protocolos como PCI, USB, I2C, etc
- Sistema de archivos virtual
 - Soportado por drivers del file system (FAT32, EXT4, etc)
 - Soportado por los character devices como por ejemplo una consola virtual o un adaptador de puerto serie
 - Soportados por protocolos como PCI, USB, I2C, etc

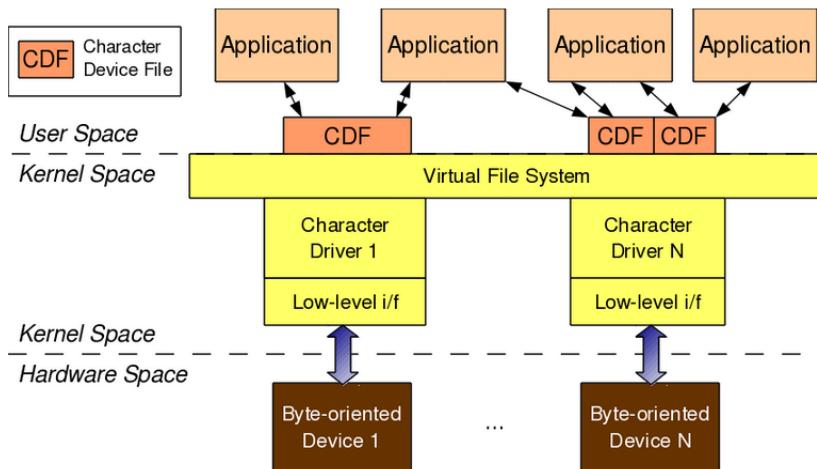


4. Character Device Drivers

Nos enfocamos en el estudio de estos. Los CDD pueden ser controladores de puertos serie, de audio, de cámara, entre otros. En el directorio /dev podemos encontrar archivos asociados a los diversos dispositivos de hardware presentes en el sistema. Podemos tomar a modo de ejemplo una de las líneas ttyUSB, que vendría a ser un character device file.

Un **character device driver** consiste en una aplicación, por ejemplo Putty, que interactúa con un Character device File (CDF) como el **ttyUSB**. Este archivo interactúa con el espacio del Kernel, donde “alguien” (el character driver) va a abrir un archivo y leer su contenido, estando esta funcionalidad soportada por el driver a través de interfaces que provee el Kernel.

cation channel between a userspace application and a character device driver. A userspace application performs usual file operations (e.g. open, read, write, close) on the character device file. Those operations are then mapped into the entry points in the linked character device driver by the virtual filesystem (VFS). Those entry points finally perform low-level accesses to the actual device to get the desired results. [10.]



5. Primer Character Device Driver

Cualquier driver de Linux consta de:

- Constructor: es llamado cada vez que **insmod** carga el módulo en el núcleo
- Destructor: es llamado cada vez que **rmmod** descarga el módulo del núcleo

Recordemos que dentro del código de un módulo se implementan las funciones **module_init()** y **module_exit()** incluidas con **module.h**.

Vínculo Aplicación - CDF → mediante el nombre del CDF

Vínculo CDF- CDDr → mediante número de CDF o índice conocido como par **<major, minor>** que identifica a un driver y se asocia a un archivo.

Este par podemos verificarlo mediante cualquiera de los siguientes comandos:

- **ls -l /dev/ | grep “^c”**

```

crw--w---- 1 root  tty      4, 12 jun 10 13:02 ttym12
crw--w---- 1 root  tty      4, 13 jun 10 13:02 ttym13
crw--w---- 1 root  tty      4, 14 jun 10 13:02 ttym14
crw--w---- 1 root  tty      4, 15 jun 10 13:02 ttym15
crw--w---- 1 root  tty      4, 16 jun 10 13:02 ttym16
crw--w---- 1 root  tty      4, 17 jun 10 13:02 ttym17
crw--w---- 1 root  tty      4, 18 jun 10 13:02 ttym18
crw--w---- 1 root  tty      4, 19 jun 10 13:02 ttym19
crw--w---- 1 franco tty    4, 20 jun 10 13:02 ttym2
crw--w---- 1 root  tty      4, 21 jun 10 13:02 ttym21
crw--w---- 1 root  tty      4, 22 jun 10 13:02 ttym22
crw--w---- 1 root  tty      4, 23 jun 10 13:02 ttym23
crw--w---- 1 root  tty      4, 24 jun 10 13:02 ttym24
crw--w---- 1 root  tty      4, 25 jun 10 13:02 ttym25
crw--w---- 1 root  tty      4, 26 jun 10 13:02 ttym26
crw--w---- 1 root  tty      4, 27 jun 10 13:02 ttym27
crw--w---- 1 root  tty      4, 28 jun 10 13:02 ttym28
crw--w---- 1 root  tty      4, 29 jun 10 13:02 ttym29
crw--w---- 1 root  tty      4, 30 jun 10 13:02 ttym30
crw--w---- 1 root  tty      4, 31 jun 10 13:02 ttym31
crw--w---- 1 root  tty      4, 32 jun 10 13:02 ttym32
crw--w---- 1 root  tty      4, 33 jun 10 13:02 ttym33
crw--w---- 1 root  tty      4, 34 jun 10 13:02 ttym34
crw--w---- 1 root  tty      4, 35 jun 10 13:02 ttym35
crw--w---- 1 root  tty      4, 36 jun 10 13:02 ttym36
crw--w---- 1 root  tty      4, 37 jun 10 13:02 ttym37
crw--w---- 1 root  tty      4, 38 jun 10 13:02 ttym38
crw--w---- 1 root  tty      4, 39 jun 10 13:02 ttym39
crw--w---- 1 root  tty      4, 40 jun 10 13:02 ttym40
crw--w---- 1 root  tty      4, 41 jun 10 13:02 ttym41

```

- **cat /proc/devices** (solo muestra el mayor)

```

franco ~ cat /proc/devices
Character devices:
 1 mem
 4 /dev/vc/0
 4 tty
 4 ttys
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
 5 ttyprintk
 6 lp
 7 vcs
10 misc
13 input
21 sg
29 fb
81 video4linux
89 i2c
99 ppdev
108 ppp
116 alsa

```

- **ls -la /dev/hda? /dev/ttys?**

```

franco ~ ls -la /dev/hda? /dev/ttys?
ls: cannot access '/dev/hda?': No such file or directory
crw-rw--- 1 root dialout 4, 64 jun 10 13:02 /dev/ttys0
crw-rw--- 1 root dialout 4, 65 jun 10 13:02 /dev/ttys1
crw-rw--- 1 root dialout 4, 66 jun 10 13:02 /dev/ttys2
crw-rw--- 1 root dialout 4, 67 jun 10 13:02 /dev/ttys3
crw-rw--- 1 root dialout 4, 68 jun 10 13:02 /dev/ttys4
crw-rw--- 1 root dialout 4, 69 jun 10 13:02 /dev/ttys5
crw-rw--- 1 root dialout 4, 70 jun 10 13:02 /dev/ttys6
crw-rw--- 1 root dialout 4, 71 jun 10 13:02 /dev/ttys7
crw-rw--- 1 root dialout 4, 72 jun 10 13:02 /dev/ttys8
crw-rw--- 1 root dialout 4, 73 jun 10 13:02 /dev/ttys9

```

6. Análisis del source “drv2.c”

Revisando el fuente drv2.c del repositorio que fue proporcionado econtramos que se utiliza una función “`alloc_chrdev_region(&first, 0, 3, "SdeC_Driver2")`”, investigamos lo que hace la misma.

Name	Arguments
<code>alloc_chrdev_region</code> — register a range of char device numbers	<i>dev_t * dev</i> output parameter for first assigned number <i>unsigned baseminor</i> first of the requested range of minor numbers
Synopsis	<i>unsigned count</i> the number of minor numbers required <i>const char * name</i> the name of the associated device or driver
Description	Allocates a range of char device numbers. The major number will be chosen dynamically, and returned (along with the first minor number) in <i>dev</i> . Returns zero or a negative error code.

Ambas funciones estan definidas en el encabezado del nucleo `<linux/fs.h>`

La función nos permitirá reservar del kernel de linux un par `<major, minor>`, y nos lo va a devolver en “`&first`” que es una referencia a una estructura del tipo `dev_t` que permite contener al par major y minor para posteriormente imprimirlos usando las macros `MAJOR` y `MINOR`.

Recursos para `<mayor, menor>` del kernel 2.6 en adelante

Tipos especiales: (definido en la cabecera del kernel `<linux/types.h>`)

- `dev_t` // contiene ambos números major & minor

Macros específicas : (definidas en la cabecera del kernel `<linux/kdev_t.h>`)

- `MAJOR(dev_t dev)` // extrae el número mayor contenido en *dev*
- `MINOR(dev_t dev)` // extrae el número menor contenido en *dev*
- `MKDEV(int major, int minor)` // crea *dev* a partir de `major & minor`

Una alternativa al uso de esta función es usar `register_chrdev_region`, sin embargo esta no se usa en el source drv2.c, pero aqui de presentan los detalles de la misma:

Name

```
register_chrdev_region — register a range of device numbers
```

Synopsis

```
int register_chrdev_region ( dev_t from,
                            unsigned count,
                            const char * name);
```

Description

Return value is zero on success, a negative error code on failure.

Arguments

dev_t from

the first in the desired range of device numbers; must include the major number.

unsigned count

the number of consecutive device numbers required

*const char * name*

the name of the device or driver.

Teniendo en cuenta todo esto, pasamos a compilar y cargar el módulo:

```
franco ➜ $ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 ➤ make
make -C /lib/modules/5.19.0-43-generic/build M=/home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 modules
make[1]: Entering directory '/usr/src/linux-headers-5.19.0-43-generic'
warning: the compiler differs from the one used to build the kernel
The kernel was built by: x86_64-linux-gnu-gcc (Ubuntu 11.3.0-1ubuntu1-22.04.1) 11.3.0
You are using:           gcc (Ubuntu 11.3.0-1ubuntu1-22.04.1) 11.3.0
CC [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2/drv2.o
MODPOST /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2/Module.symvers
CC [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2/drv2.mod.o
LD [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2/drv2.ko
BTB [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2/drv2.ko
Skipping BTF generation for /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2/drv2.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.19.0-43-generic'
franco ➜ $ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 ➤ sudo insmod drv2.ko
franco ➜ $ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 ➤ sudo insmod drv2.o
[sudo] password for franco:
franco ➜ $ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 ➤
```

Verificamos que se haya cargado con **lsmod**

```
franco ➜ $ lsmod
Module           Size  Used by
drv2            16384  0
vboxnetadp      28672  0
vboxnetflt      28672  0
vboxdrv        573440  2 vboxnetadp,vboxnetflt
rfcomm          86016   4
```

Revisamos el log del kernel con **dmesg** y podemos observar el par que se imprimió

```
[200000.019100] audit: type=1520 audit(1000/01/04 22.02): audid=1000 uid=1000 gid=1000 ses=3 subj=s
nm="pool-org.gnome." exe="/bin/snap-store" sig=0 arch=c000003e syscall=93 compat=0 ip=0x7f19d8b7ec4b
[297847.992622] SdeC_drv2 Registrado exitosamente...
[297847.992629] <Major, Minor>: <508, 0>
franco ➜ $
```

Ahora usamos **cat /proc/devices** para verificar que el mayor del drv2 es 508

```
250 dax
251 dimmctl
252 ndctl
253 tpm
254 gpiochip
508 SdeC Driver2
509 kfd
510 aux
511 cec
```

Ahora descargamos el módulo teniendo en cuenta que el destructor en el fuente hace uso de la función **unregister_chrdev_region(first, 3)**

Investigamos un poco esta función: el parámetro 3 es el número de números de dispositivo cuyo registro se debe dar de baja.

Name	Synopsis
unregister_chrdev_region — return a range of device numbers	<code>void unregister_chrdev_region (dev_t from, unsigned count);</code>
Arguments	Description
<i>from</i>	This function will unregister a range of <i>count</i> device numbers, starting with <i>from</i> . The caller should normally be the one who allocated those numbers in the first place...
<i>count</i>	
the first in the range of numbers to unregister	
<i>count</i>	
the number of device numbers to unregister	

Ejecutamos rmmod drv2.ko

```
franco ~ main ?:13 ~/labs/siscomp/lab5/device_drivers/device-drivers/FuentesDrv2 sudo rmmod drv2.ko
[sudo] password for franco:
franco ~ main ?:13 ~/labs/siscomp/lab5/device_drivers/device-drivers/FuentesDrv2 dmesg
dmesg: read kernel buffer failed: Operation not permitted
franco ~ main ?:13 ~/labs/siscomp/lab5/device_drivers/device-drivers/FuentesDrv2 1 sudo dmesg
```

Observamos que en el log del Kernel ahora figura el mensaje configurado en el source para cuando se descarga el módulo:

```
[259280.006201] audit: type=1326 audit(1686672242.907:201): auid=1000 uid=1000 gid=1000 ses=3 subj=snap.snap-store.ubuntu-software pid=6023 co
mm="pool-org.gnome." exe="/bin/snap-store" sig=0 arch=0x00003e syscall=93 compat=0 ip=0x7f19d8b7ec4b code=0x50000
[288086.019188] audit: type=1326 audit(1686701042.921:202): auid=1000 uid=1000 gid=1000 ses=3 subj=snap.snap-store.ubuntu-software pid=6023 co
mm="pool-org.gnome." exe="/bin/snap-store" sig=0 arch=0x00003e syscall=93 compat=0 ip=0x7f19d8b7ec4b code=0x50000
[297847.992622] [SdeCdrv2 Registrado exitosamente..!]
[297847.992629] <Major, Minor: <508, 0>
[300143.757200] [drm:dc_dmub_srv wait idle [amdgpu]] *ERROR* Error waiting for DMUB idle: status=3
[320494.343613] audit: type=1326 audit(1686733451.242:203): auid=1000 uid=1000 gid=1000 ses=3 subj=snap.snap-store.ubuntu-software pid=6023 co
mm="pool-org.gnome." exe="/bin/snap-store" sig=0 arch=0x00003e syscall=93 compat=0 ip=0x7f19d8b7ec4b code=0x50000
[336484.161794] [SdeCdrv2 dice Adios mundo cruel..!]
franco ~ main ?:13 ~/labs/siscomp/lab5/device_drivers/device-drivers/FuentesDrv2
```

Ahora que ya hemos mostrado lo que sucede al cargar y descargar el módulo, volvemos a cargarlo de la misma forma que se hizo previamente con el objetivo de ahora crear manualmente un **CDF** que asocie nuestro Device Driver “**drv2**” (que ya hemos visto que tiene asignado el número major **508**) con un número **minor**. De esta forma por cada CDF que se crea, vamos a poder asociar un nuevo dispositivo a un driver en particular.

Primero comprobamos que hasta el momento en **/dev** no existe ningun dispositivo que tenga asociado el major 508, para esto usamos el comando **ls /dev -la | grep 508**.

```
franco ~ 1 ls /dev -la | grep 509
crw-rw---- 1 root render 509, 0 jun 10 13:02 kfd
franco ~ ls /dev -la | grep 508
franco ~ 1
```

Esto comprueba que no existe de momento ningún dispositivo creado bajo /dev con el mayor 508.

Para crear los archivos y asociarlos podemos usar el comando **mknod**:

```
sudo mknod /dev/drv2_0 c MMM 0
sudo mknod /dev/drv2_1 c MMM 1
```

Donde para nuestro caso reemplazamos MMM = 508. Tener en cuenta que los minor en realidad no hace falta que sean consecutivos.

```
franco ~ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 1 sudo mknod /dev/drv2_0 c 508 0
franco ~ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 1 sudo mknod /dev/drv2_1 c 508 1
```

Luego de ejecutar los comandos volvemos a revisar **/dev**. Podemos observar que ahora existen los dos archivos que hemos creado y están asociados al mayor 508

```
franco ~ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 ls /dev -la | grep 508
crw-r--r-- 1 root root 508, 0 jun 14 11:16 drv2_0
crw-r--r-- 1 root root 508, 1 jun 14 11:16 drv2_1
franco ~ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 1
```

Dado que se trata de un character device, en principio deberíamos poder hacer **cat /dev/drv2_0**

```
franco ~ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 cat /dev/drv2_0
cat: /dev/drv2_0: No such device or address
franco ~ main ?:13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv2 1
```

Pero vemos que se produce un error, esto se debe a que cat hace un read sobre el SO, pero en nuestro fuente **drv2.c** en ninguna parte hemos especificado que se hace cuando llega un read. En este driver falta implementar las funciones de read y write. Esto se encuentra implementado en el fuente **drv3.c**.

7. Análisis del source “drv3.c”

Este source implementa la creación automática de los CDF.

Tener en cuenta

“Hasta el kernel 2.4, la creación de los archivos de dispositivo fue realizada automáticamente por el kernel mismo, llamando a las API apropiadas de devfs.

A medida que el kernel evolucionó, los desarrolladores se dieron cuenta de que los DF son más una cuestión de espacio de usuario y, por lo tanto, como política, solo los usuarios deberían tratarlo, y no el kernel.

Con esta idea, ahora el núcleo solo completa, para el dispositivo en consideración, la clase de dispositivo y la información del dispositivo apropiadas en /sys/class

Luego, el espacio de usuario necesita interpretarlo y tomar una acción apropiada. En la mayoría de los sistemas de escritorio Linux, el demonio udev recoge esa información y, en consecuencia, crea los archivos de dispositivo."

Creación automática de los CDF

La clase de dispositivo se crea de la siguiente manera:

```
struct class *cl=class_create(THIS_MODULE, "<device class name>");
```

y luego la información del dispositivo (<major, minor>) se completa en esta clase con:

```
device_create(cl, NULL, first, NULL, "<device name format>", ...);
```

donde **first** es tipo **dev_t** con el correspondiente <major, minor>.

Las llamadas complementarias o inversas correspondientes, que deben llamarse en orden cronológicamente inverso, son las siguientes:

```
device_destroy(cl, first);
```

```
class_destroy(cl);
```

```
if (IS_ERR(cl = class_create(THIS_MODULE, "chardrv")))
{
    unregister_chrdev_region(first, 1);
    return PTR_ERR(cl);
}

// Ese puntero "cl" será utilizado en la llamada a device_create, junto a los valores de
// MAJOR y MINOR contenidos en "first" y el nombre que se le dará a esta nueva entrada.

if (IS_ERR(dev_ret = device_create(cl, NULL, first, NULL, "SdeC_drv3")))
{
    class_destroy(cl);
    unregister_chrdev_region(first, 1);
    return PTR_ERR(dev_ret);
}
```

Se crea una clase “**chardrv**” y un dispositivo dentro de esa clase “**SdeC_drv3**”

También podemos ver que, a diferencia del anterior, este source implementa las funciones de read y write.

```

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
};

cdev_init(&c_dev, &pugs_fops);
if ((ret = cdev_add(&c_dev, first, 1)) < 0)
{
    device_destroy(cl, first);
    class_destroy(cl);
    unregister_chrdev_region(first, 1);
    return ret;
}
return 0;
}

```

Comenzando compilando y cargando el módulo:

```

franco ~ main ?13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 ls
drv3.c Makefile
franco ~ main ?13 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 make
make -C /lib/modules/5.19.0-43-generic M=/home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 modules
make[1]: Entering directory '/usr/src/linux-headers-5.19.0-43-generic'
warning: the compiler differs from the one used to build the kernel
The kernel was built by: x86_64-linux-gnu-gcc (Ubuntu 11.3.0-1ubuntu1-22.04.1) 11.3.0
You are using:          gcc (Ubuntu 11.3.0-1ubuntu1-22.04.1) 11.3.0
CC [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3/drv3.o
MODPOST /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3/Module.symvers
CC [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3/drv3.mod.o
LD [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3/drv3.ko
BTF [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3/drv3.ko
Skipping BTF generation for /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3/drv3.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.19.0-43-generic'
franco ~ main ?26 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 ls
drv3.c drv3.ko drv3.mod drv3.mod.c drv3.mod.o Makefile modules.order Module.symvers
franco ~ main ?26 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3

franco ~ main ?26 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 sudo insmod drv3.ko
[sudo] password for franco:
franco ~ main ?26 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 sudo dmesg

[337535.778345] <Major, Minor>: <508, 0>
[340135.557992] SdeC drv3 Registrado exitosamente..!!

```

Usamos el comando `find /sys -name Sde*` para encontrar la carpeta donde se encuentra el archivo que supuestamente se creó automáticamente.

```

franco ~ main ?26 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 sudo find /sys -name Sde*
/sys/class/chardrv/SdeC_drv3
/sys/devices/virtual/chardrv/SdeC_drv3
franco ~ main ?26 ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3

```

- Que es sys/class?

Se trata de un subdirectorio de el sistema de archivos sysfs, que es un pseudo sistema de archivos (filesystem virtual) que provee una interfaz a estructuras de datos del kernel. sysfs se encuentra montado típicamente como /sys. El subdirectorio /sys/class contiene una sola capa de subdirectorios para cada una de las device classes que han sido registradas en el sistema. Fuente: [sysfs\(5\) - Linux manual page \(man7.org\)](https://man7.org/linux/man-pages/man5/sysfs.5.html)

- ¿Para que sirven las clases ?

Las clases en sysfs sirven para organizar y representar jerárquicamente los dispositivos del sistema según su tipo o clase, proporcionando una interfaz para acceder a la información y configuraciones relacionadas con esos dispositivos.

Verificamos que efectivamente se ha creado “**SdeC_drv3**” en **/sys/class/chardrv**

```
franco ➤ main ?:26 ➤ ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 ➤ ls /sys/class/chardrv
{deC_drv3}
```

También se ha creado un directorio “**SdeC_drv3**” en **/sys/devices/virtual/chardrv**

```
franco ➤ ~ ➤ ls /sys/devices/virtual/chardrv -l
total 0
drwxr-xr-x 3 root root 0 jun 14 11:31 SdeC_drv3
franco ➤ ~ ➤ ls /sys/devices/virtual/chardrv/SdeC_drv3/
dev power subsystem uevent
franco ➤ ~ ➤
```

Podemos hacer read/write a estos CDF con **ls -al /dev/SdeC_drv3**

```
franco ➤ main ?:26 ➤ ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 ➤ ls -al /dev/SdeC_drv3
crw----- 1 root root 507, 0 jun 14 11:31 /dev/SdeC_drv3
franco ➤ main ?:26 ➤ ~/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv3 ➤
```

Ahora revisamos /proc/devices y vemos que tiene asignado el major **507**

```
253 tpmm
254 gpiochip
507 SdeC_drv3
508 SdeC_Driver2
509 kfd
510 aux
511 cec
```

Decodificando las operaciones del CDF

```
root@franco-lenovo:/home/franco# cat /dev/SdeC_drv3
root@franco-lenovo:/home/franco# echo "Hola driver.." > /dev/SdeC_drv3
root@franco-lenovo:/home/franco# dmesg
```

```
[343438.240434] Driver3_SdeC: open()
[343438.240451] Driver3_SdeC: read()
[343438.240462] Driver3_SdeC: close()
[343462.916982] Driver3_SdeC: open()
[343462.917021] Driver3_SdeC: write()
[343462.917036] Driver3_SdeC: close()
root@franco-lenovo:/home/franco# 
```

Vemos que al hacer **cat** se generaron los open, read y close y que al hacer el **echo** se generaron open, write y close.

8. Trabajamos con el source “drv4.c”

make e insmod

```
franco ➤ main ?:26 ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesDrv4 ➤ make
make -C /lib/modules/5.19.0-43-generic/build M=/home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv4 modules
make[1]: Entering directory '/usr/src/linux-headers-5.19.0-43-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
  You are using:          gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
CC [M]  /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv4/drv4.o
MODPOST /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv4/Module.symvers
CC [M]  /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv4/drv4.mod.o
LD [M]  /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv4/drv4.ko
BTF [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv4/drv4.ko
Skipping BTF generation for /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesDrv4/drv4.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.19.0-43-generic'
franco ➤ main ?:39 ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesDrv4 ➤ ls
drv4.c  drv4.ko  drv4.mod  drv4.mod.o  drv4.o  Makefile  modules.order  Module.symvers
franco ➤ main ?:39 ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesDrv4 ➤ sudo insmod drv4.ko
```

lsmod

```
franco ➤ main ?:39 ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesDrv4 ➤ lsmod | grep drv4
drv4           16384  0
```

dmesg

```
[344330.282924] SdeC_drv4: Registrado exitosamente...!
franco ➤ main ?:39 ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesDrv4 ➤
```

cat

```
root@franco-lenovo:/home/franco# cat /dev/SdeC_drv4
root@franco-lenovo:/home/franco# 
```

lsmod

```
[344497.345329] SdeC_drv4: open()
[344497.345346] SdeC_drv4: read()
[344497.345361] SdeC_drv4: read()
[344497.345373] SdeC_drv4: close()
```

echo

```
root@franco-lenovo:/home/franco# echo -n "H" > /dev/SdeC_drv4
root@franco-lenovo:/home/franco# 
```

lsmod

```
[344497.345329] SdeC_drv4: open()
[344497.345346] SdeC_drv4: read()
[344497.345361] SdeC_drv4: read()
[344497.345373] SdeC_drv4: close()
[344595.237033] SdeC_drv4: open()
[344595.237052] SdeC_drv4: write()
[344595.237069] SdeC_drv4: close()
```

A diferencia de lo que ocurría con drv3, con drv4 se generan dos lecturas al usar el comando **cat**.

9. Módulo “Clipboard” /proc

Recordemos que /proc es un filesystem virtual (como el /sys) que se crea con el kernel y tiene cosas como cpubinfom, meminfo, etc.

Podemos crear un módulo de kernel que genere un archivo en **/proc**, primero revisamos el código del **clipboard.c**

```
int init_clipboard_module( void ) #define BUFFER_LENGTH PAGE_SIZE
{
    int ret = 0;           se aloca un trozo de memoria del tamaño de una página
    clipboard = (char *)vmalloc( BUFFER_LENGTH );
    if (!clipboard) {
        ret = -ENOMEM;
    } else {
        memset( clipboard, 0, BUFFER_LENGTH );      crea la entrada en /proc
        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;
}
```

```
static const struct proc_ops proc_entry_fops = {
    .proc_read = clipboard_read,    operaciones
                                    esenciales
    .proc_write = clipboard_write,
```

Make, insmod

```
franco ➤ ⚡ main ?:39 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ make
make -C /lib/modules/5.19.0-43-generic/build M=/home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesClipboard modules
make[1]: Entering directory '/usr/src/linux-headers-5.19.0-43-generic'
warning: the compiler differs from the one used to build the kernel
  The kernel was built by: x86_64-linux-gnu-gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
  You are using:          gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
  CC [M]  /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesClipboard/clipboard.o
  MODPOST /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesClipboard/Module.symvers
  CC [M]  /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesClipboard/clipboard.mod.o
  LD [M]  /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesClipboard/clipboard.ko
  BTF [M] /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesClipboard/clipboard.ko
Skipping BTF generation for /home/franco/labs_siscomp/lab5_device_drivers/device-drivers/FuentesClipboard/clipboard.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.19.0-43-generic'
franco ➤ ⚡ main ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ ls
clipboard.c clipboard.ko clipboard.mod clipboard.mod.c clipboard.o Makefile modules.order Module.symvers
franco ➤ ⚡ main ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ sudo insmod clipboard.ko
[sudo] password for franco:
franco ➤ ⚡ main ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ █
```

lsmod

```
franco ➤ ⚡ main ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ lsmod | grep clipboard
clipboard           16384  0
franco ➤ ⚡ main ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ █
```

dmesg

```
[345120.428810] Clipboard: Modulo cargado..!!
```

Inspeccionamos /proc, y vemos que se creó el archivo

```
franco ➤ ⚡ main ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ ls /proc -la | grep clipboard
-rw-rw-rw-  1 root      root          0 jun 14 12:56 clipboard
franco ➤ ⚡ main ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ █
```

Tener en cuenta que en este caso no se trata de un driver, sino de una entrada en /proc, por lo que es como si fuera un archivo.

echo y cat

```
franco ➤ ⚡ main U:1 ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ 1 echo "hola que tal" >| /proc/clipboard
franco ➤ ⚡ main U:1 ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ cat /proc/clipboard
hola que tal
franco ➤ ⚡ main U:1 ?:52 ➤ ~/labs siscomp/lab5 device_drivers/device-drivers/FuentesClipboard ➤ █
```

De esta forma hemos llevado texto desde el espacio de usuario al espacio del kernel (con **echo**) y lo hemos traído de vuelta al espacio de usuario (con **cat**)

10. Desafío sobre Raspberry Pi

Leer GPIO desde el Kernel, traerlo al espacio de usuario y en el espacio de usuario graficar lo que esté ocurriendo.

Para superar este TP tendrán que:

- Diseñar y construir un CDD que permita sensar dos señales externas con un periodo de UN segundo.
- Luego una aplicación a nivel de usuario deberá leer UNA de las dos señales y graficarla en función del tiempo. La aplicación también debe poder indicarle al CDD cuál de las dos señales leer. Las correcciones de escalas de las mediciones, de ser necesario, se harán a nivel de usuario. Los gráficos de la señal deben indicar el tipo de señal que se está sensando, unidades en abscisas y tiempo en ordenadas. Cuando se cambie de señal el gráfico se debe "resetear" y acomodar a la nueva medición.

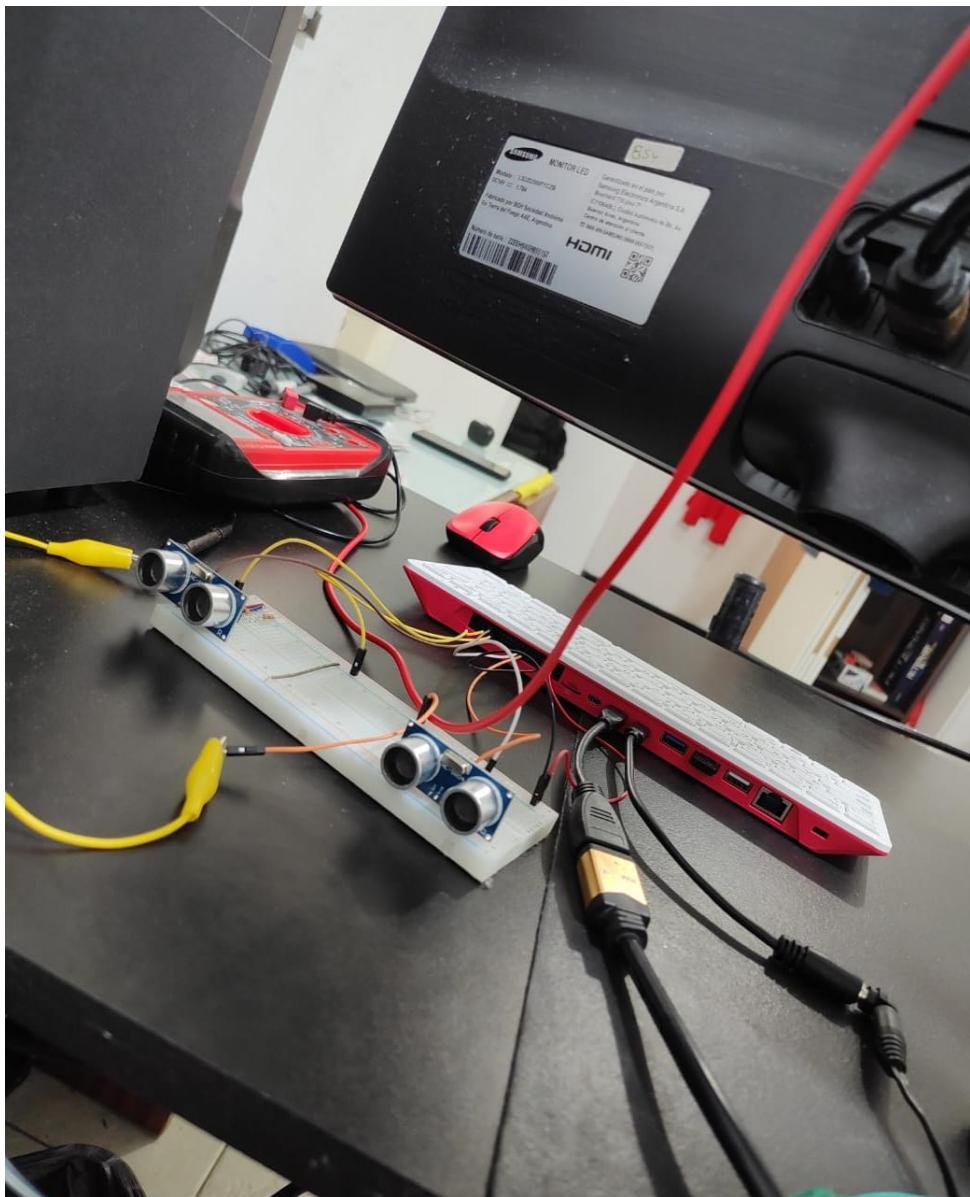
Se recomienda utilizar una Raspberry Pi para desarrollar este TP.



Due to the flexibility of GPIO pins, they are often used in a wide range of integrated circuits (IC), SoC, embedded platforms such as Arduino, BeagleBoard, Stellaris Launchpad, and Raspberry Pi, multiple function chips, for example audio codes and video cards, or Programmable Logic Devices (PLD) [5]. It should be noted that GPIO pins 2 and 3 on Raspberry Pi are initially configured as I2C interfaces with 1.8KΩ pull-up resistors, and pin 15 being configured as a serial UART receive pin. When Raspberry Pi is booting, these pins must be left unused. Otherwise, the board will not be able to boot successfully.

```
struct file_operations {  
    struct module *owner;  
    loff_t (*llseek) (struct file *, loff_t, int);  
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);  
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);  
    int (*open) (struct inode *, struct file *);  
    int (*release) (struct inode *, struct file *);  
};
```

Listing 1. Entry points for the GPIO device driver



Repo: <https://github.com/francoriba/LinuxPiDriverDev>