

# Sistemas Operativos

Universidad Complutense de Madrid  
2020-2021

## Módulo 4: Gestión de Entrada/Salida

Juan Carlos Sáez

# Contenido

## 1 Introducción

- Drivers

## 2 Gestión de E/S en Linux

## 3 Módulos del kernel Linux

## 4 Drivers de dispositivos de caracteres

## 5 Práctica 4

# Contenido

## 1 Introducción

- Drivers

## 2 Gestión de E/S en Linux

## 3 Módulos del kernel Linux

## 4 Drivers de dispositivos de caracteres

## 5 Práctica 4

# Introducción

- El corazón de una computadora lo constituye la CPU, pero no serviría de nada sin:
  - Dispositivos de almacenamiento no volátil
    - Secundario: discos
    - Terciario: cintas
  - Dispositivos periféricos que le permitan interactuar con el usuario (los teclados, ratones, micrófonos, cámaras, etc.)
  - Dispositivos de comunicaciones: permiten conectar a la computadora con otras a través de una red

*Dispositivos de salida:  
Impresora, monitor, ...*



*Unidad principal (UCP, registros, memoria RAM, Entrada/Salida (discos internos, red,...))*

*Dispositivos de entrada  
(teclado, ratón)*

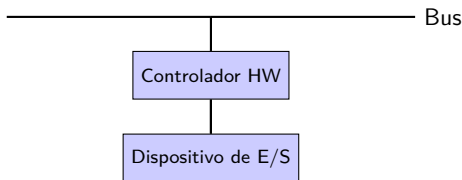


*Dispositivos de E/S  
(discos, cintas, modem, ...)*



# Conexión de dispositivos de E/S

- En el sistema de E/S de un computador se distinguen dos elementos:
  - 1 **Periféricos o dispositivos de E/S:** elementos que se conectan a la unidad central de proceso (CPU) a través de controladores hardware.
    - Son el componente mecánico que se conecta al computador.
  - 2 **Controladores HW de dispositivos o unidades de E/S:** Se encargan de hacer la transferencia de información entre la memoria principal y los periféricos.
    - Son el componente electrónico a través del cual se conecta el dispositivo de E/S.
    - Tienen una conexión al bus de la computadora y otra para el dispositivo
    - **Advertencia:** Controlador HW  $\neq$  Controlador SW o *Driver*



# Objetivos del software de E/S del SO

- **Facilitar el manejo de los dispositivos periféricos.**
  - Ofrecer una interfaz entre los dispositivos y el resto del sistema que sea sencilla y fácil de utilizar
- **Permitir la conexión de nuevos dispositivos** sin que sea necesario remodelar por completo el software de E/S del SO
  - Clases de dispositivos
- **Optimizar la E/S del sistema**, proporcionando mecanismos de incremento de prestaciones donde sea necesario
- **Automatizar la instalación de nuevos dispositivos de E/S**, usando mecanismos del tipo *plug & play*

# Drivers (I)

- **Driver:** componente software del SO destinado a gestionar un tipo específico de dispositivo de E/S
  - También llamado **controlador SW o manejador de dispositivo**
- Cada driver se divide en dos partes:
  - 1 **Código independiente del dispositivo:** para dotar al nivel superior del SO de una interfaz
    - Interfaz similar para acceso a dispositivos muy diferentes
    - Simplifica la labor de portar SSOOs y aplicaciones a nuevas plataformas hardware
  - 2 **Código dependiente del dispositivo:** necesario para interactuar con dispositivo de E/S a bajo nivel
    - Interacción directa con el controlador HW
    - Gestión de interrupciones
    - ...

## Drivers (II)

### Otras características de los disp. E/S

- Necesario considerar características inherentes de los disp. de E/S
  - Algunos dispositivos soportan acceso a nivel de byte pero otros no...
  - Ejemplo: No podemos acceder a nivel de byte a un disco (*acceso a nivel de bloque*)

### En general, los dispositivos se dividen en dos clases:

- 1 Dispositivos de bloque
  - disco, red, pantalla, ...
- 2 Dispositivos de carácter
  - teclado, ratón, puerto serie, ...



# Contenido

## 1 Introducción

- Drivers

## 2 Gestión de E/S en Linux

## 3 Módulos del kernel Linux

## 4 Drivers de dispositivos de caracteres

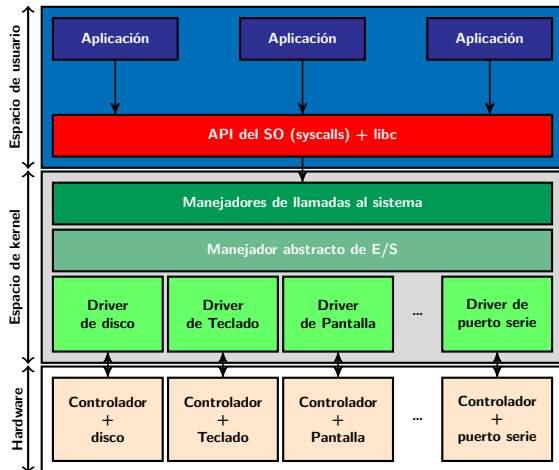
## 5 Práctica 4

# Gestión de E/S en Linux

- Casi todos los **dispositivos de E/S se representan como ficheros especiales** (que pueden ser de bloque o carácter)
  - `/dev/sda1` para la primera partición del primer disco SATA o USB
  - `/dev/ttyS0` para el primer puerto serie
  - `/dev/tty0` para el primer terminal/consola de texto
  - `/dev/lp0` para la impresora
- El **acceso a estos ficheros especiales** se realiza mediante las llamadas al sistema `open()`, `read()`, `write()` y `close()`
  - Un programa de usuario puede acceder al dispositivo de E/S, siempre y cuando usuario tenga permisos de acceso al fichero especial
  - Excepcionalmente puede requerirse `ioctl()` para realizar operaciones de control
    - `man 2 ioctl`

# Visión general

- Cada fichero de dispositivo tiene un **driver** asociado



Cuando un programa de usuario invoca una llamada al sistema sobre un fichero especial (p.ej., `read()`), se ejecuta una función del driver que hace el trabajo correspondiente

## Major and minor number

- Cada fichero de dispositivo tiene asociado un **par (*major*, *minor*)** que lo identifica de forma unívoca
  - *major*: ID de la clase de dispositivo
  - *minor*: ID local para que driver pueda distinguir al dispositivo en caso de gestionar varios
- En Linux, los disp. de E/S se agrupan en *clases*:
  - Cada clase de dispositivo tiene un **major number** (ID) asociado
  - <https://www.kernel.org/doc/Documentation/admin-guide/devices.txt>
- Ejemplo: Driver que gestiona 2 discos duros
  - Discos representados mediante ficheros de dispositivo `/dev/sda` y `/dev/sdb`
  - Ambos ficheros especiales tendrán el mismo *major number* pero distinto *minor number*
- Dos drivers podrían tener el mismo *major* asignado, pero trabajan con distintos rangos de *minor numbers*
  - Esto permite al kernel saber qué driver debe encargarse de gestionar cada dispositivo

## Major and minor number

- El comando `stat` permite consultar el tipo de dispositivo (fichero especial) así como el mayor y menor number asociado al mismo

terminal

```
osuser@debian:~$ stat /dev/tty1
```

```
File: </dev/tty1>
Size: 0      Blocks: 0      IO Block: 4096   character special file
Device: 5h/5d  Inode: 1259      Links: 1      Device type: 4,1
Access: (0600/crw-----)  Uid: (   0/   root)   Gid: (   0/   root)
Access: 2016-02-24 08:18:59.663968939 +0100
Modify: 2016-02-24 08:18:59.523954207 +0100
Change: 2016-02-24 08:18:59.523954207 +0100
Birth: -
```

```
osuser@debian:~$ stat /dev/sda1
```

```
File: </dev/sda1>
Size: 0      Blocks: 0      IO Block: 4096   block special file
Device: 5h/5d  Inode: 1708      Links: 1      Device type: 8,1
Access: (0660/brw-rw----)  Uid: (   0/   root)   Gid: (   6/   disk)
Access: 2016-02-24 08:18:59.663968939 +0100
Modify: 2016-02-24 08:18:59.523954207 +0100
Change: 2016-02-24 08:18:59.523954207 +0100
Birth: -
```

# Drivers y módulos del kernel Linux

## terminal

```
osuser@debian:~$ cat /proc/devices
```

```
Character devices:
```

```
1 mem
4 /dev/vc/0
4 tty
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
7 vcs
10 misc
13 input
21 sg
29 fb
108 ppp
128 ptm
136 pts
180 usb
189 usb_device
226 drm
...
```

```
Block devices:
```

```
259 blkext
8 sd
11 sr
65 sd
66 sd
67 sd
68 sd
69 sd
...
```

- La asociación entre el *driver* del dispositivo y el número de versión mayor asignado puede consultarse en `/proc/devices`
- La mayor parte de los *drivers* en Linux se implementan como módulos cargables del kernel
  - Implementan una interfaz especial
  - Se registran a sí mismos como *driver* de caracteres o bloques

# Contenido

## 1 Introducción

- Drivers

## 2 Gestión de E/S en Linux

## 3 Módulos del kernel Linux

## 4 Drivers de dispositivos de caracteres

## 5 Práctica 4

# Módulos cargables del kernel Linux (I)

## ¿Qué es un módulo cargable?

- Un “fragmento de código” que puede cargarse/descargarse en el mapa de memoria del SO (kernel) bajo demanda
- Sus funciones se ejecutan en modo kernel (privilegiado)
  - **Cualquier error fatal en el código “cuelga” el SO**
  - Herramientas de depuración menos elaboradas
    - `printk()`: Imprimir mensajes en fichero de log del kernel
    - `dmesg` : Muestra contenido del fichero de log del kernel

También existe soporte para módulos cargables en otros sistemas tipo UNIX (BSD, Solaris) y en MS Windows



# Módulos cargables del kernel Linux (II)

## Ventajas de los módulos del kernel

- 1 Reducen el *footprint* del kernel del SO
  - Cargamos únicamente los componentes SW (módulos) necesarios
- 2 Permiten extender la funcionalidad del kernel en caliente (sin tener que reiniciar el sistema)
  - Mecanismo para implementar/desplegar *drivers*
- 3 Permiten un diseño más *modular* del sistema

# Módulos cargables del kernel Linux (III)

- Los módulos disponibles para nuestro kernel se encuentran en el directorio `/lib/modules/${KERNEL_VERSION}`
  - `KERNEL_VERSION=$(uname -r)`
- Podemos saber qué módulos están cargados con `lsmod`
  - `/proc/modules`

## Terminal

```
osuser@debian:~$ lsmod
```

Module	Size	Used by
mperf	935	0
cpufreq_stats	2139	0
bluetooth	55448	2
cpufreq_powersave	650	0
cpufreq_userspace	1464	0
cpufreq_conservative	3791	0
binfmt_misc	4994	1
uinput	5172	1
fuse	49890	3
acpiphp	12757	0
loop	10809	0
tpm_tis	5725	0
...		

## Anatomía de un módulo cargable

- En lugar de una función `main()`, el código de un módulo tiene funciones `init_module()` y `cleanup_module()`
  - `init_module()` se invoca cuando se carga el módulo del kernel
  - `cleanup_module()` se invoca al eliminar/descargar el módulo del kernel
- Al compilar el módulo se genera un fichero `.ko` que es un fichero objeto ELF (*Executable and Linkable Format*) especial

### Carga y descarga de módulos

- Para cargar el módulo se usa el comando `insmod`  
`$ insmod my_module.ko`
- Un módulo puede descargarse con `rmmod`  
`$ rmmod my_module`
- Solo el administrador (*root*) puede ejecutar ambos comandos
  - En la máquina virtual, usar `sudo <comando>` (+ password de usuario)

# Módulo del kernel (“Hello World”)

## hello.c

```
#include <linux/module.h> /* Needed by all modules */
#include <linux/kernel.h> /* Needed for KERN_INFO */

int init_module(void)
{
    printk(KERN_INFO "Hello world.\n");

    /*
     * A non 0 return means init_module failed; module can't be loaded.
     */
    return 0;
}

void cleanup_module(void)
{
    printk(KERN_INFO "Goodbye world.\n");
}
```

# Compilación de Módulos

## ■ Compilación gestionada mediante un fichero *Makefile*

- Es necesario tener instalados los ficheros de cabecera (`__headers__`) del kernel en ejecución (ya instalados en la máquina virtual)
- El *Makefile* debe estar **ubicado en el mismo directorio que las fuentes** del módulo
  - Para compilar, teclear `make`
  - “`make clean`”: borrar ficheros resultantes de la compilación
  - La ruta donde se almacena el *Makefile*/fuentes **NO** puede contener espacios

### Makefile (módulo de un solo fichero `.c`)

```
obj-m = hello.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

# Ejemplo: compilación, carga y descarga (I)

## Terminal

```
osuser@debian:~/A4Files/Hello$ ls
Makefile  hello.c
osuser@debian:~/A4Files/Hello$ make
make -C /lib/modules/4.9.111-lin/build M=/home/osuser/A4Files/Hello modules
make[1]: Entering directory `/usr/src/linux-headers-4.9.111-lin'
  CC [M]  /home/osuser/A4Files/Hello/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/osuser/A4Files/Hello/hello.mod.o
  LD [M]  /home/osuser/A4Files/Hello/hello.ko
make[1]: Leaving directory `/usr/src/linux-headers-4.9.111-lin'
osuser@debian:~/A4Files/Hello$ sudo insmod hello.ko
[sudo] password for osuser:
osuser@debian:~/A4Files/Hello$ lsmod | head
Module                Size  Used by
hello                  836   0
binfmt_misc           6160   1
uinput                6879   1
nfsd                  198017  2
auth_rpcgss           37914  1 nfsd
oid_registry          2051  1 auth_rpcgss
exportfs              3400  1 nfsd
nfs_acl               2175  1 nfsd
nfs                   152253  0
```

## Ejemplo: compilación, carga y descarga (II)

### Terminal

```
osuser@debian:~/A4Files/Hello$ sudo dmesg | tail
[ 4229.560018] usb 1-2.1: Product: Virtual Bluetooth Adapter
[ 4229.560022] usb 1-2.1: Manufacturer: VMware
[ 4229.560026] usb 1-2.1: SerialNumber: 000650268328
[ 4645.867113] hello: module license 'unspecified' taints kernel.
[ 4645.867117] Disabling lock debugging due to kernel taint
[ 4645.867748] Hello world.
osuser@debian:~/A4Files/Hello$ sudo rmmod hello
osuser@debian:~/A4Files/Hello$ sudo dmesg | tail
[ 4229.396166] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 4229.560005] usb 1-2.1: New USB device found, idVendor=0e0f, idProduct=0008
[ 4229.560013] usb 1-2.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 4229.560018] usb 1-2.1: Product: Virtual Bluetooth Adapter
[ 4229.560022] usb 1-2.1: Manufacturer: VMware
[ 4229.560026] usb 1-2.1: SerialNumber: 000650268328
[ 4645.867113] hello: module license 'unspecified' taints kernel.
[ 4645.867117] Disabling lock debugging due to kernel taint
[ 4645.867748] Hello world.
[ 4741.556845] Goodbye world.
osuser@debian:~/A4Files/Hello$ lsmod | head
Module                Size  Used by
binfmt_misc           6160   1
uinput                6879   1
nfsd                  198017  2
```

## API del kernel para módulos

- En el kernel no hay libc
  - Solo se implementan algunas funciones de la libc
- Los módulos del kernel pueden invocar funciones del kernel para asignar memoria, administrar temporizadores, etc
  - Uso avanzado del API para módulos: **Optativa “Arquitectura Interna de Linux y Android”**

### Manejo de Cadenas

strlen, sprintf, strcmp, strncmp, sscanf, strcat, memset, memcpy, strtok, ...

### Reservar y liberar memoria dinámica <linux/vmalloc.h>

```
void *vmalloc( unsigned long size );  
void vfree( void *addr );
```



## Otras funciones útiles

### Incrementar/decrementar contador de referencia del módulo

- `try_module_get(THIS_MODULE);`
- `module_put(THIS_MODULE);`

### Copia espacio usuario $\longleftrightarrow$ kernel `<linux/uaccess.h>`

- `copy_from_user()`, `copy_to_user()`, `get_user()`, `put_user()`

# Contenido

## 1 Introducción

- Drivers

## 2 Gestión de E/S en Linux

## 3 Módulos del kernel Linux

## 4 Drivers de dispositivos de caracteres

## 5 Práctica 4

# Drivers de dispositivos de caracteres

## Interfaz para drivers de dispositivos de caracteres

```
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 (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ....
};
```

# Implementación de un driver

- 1 Crear un módulo del kernel con funciones `init_module()` y `cleanup_module()`
- 2 Implementar las operaciones de la interfaz del disp. caracteres
  - `struct file_operations`
- 3 En la función de inicialización:
  - Reservar major number y rango de minor numbers para el driver
    - `alloc_chrdev_region()`
  - Crear una estructura `cdev` y asociarle las operaciones y el rango de major/minor
    - Usar `cdev_alloc()`, `cdev_init()` y `cdev_add()`
- 4 En la función de descarga:
  - Destruir estructura `cdev_t`
    - `cdev_del()`
  - Liberar el rango de (major, minor) reservado
    - `unregister_chrdev_region()`

## Representación de (major,minor) en el kernel

- En el kernel Linux el par (*major,minor*) está representado mediante el tipo `dev_t`
  - `dev_t`: número de 32 bits (12 bits major, 20 bits minor)
- Por motivos históricos el empaquetamiento es complejo, se utilizan macros:
  - Acceso a números: `MAJOR(dev_t dev)`, `MINOR(dev_t dev)`
  - Construcción de par: `MKDEV(int major, int minor)`
    - Ejemplo: `dev_t pair=MKDEV(4,1);`

# Descripción de la API (I)

## alloc\_chrdev\_region()

```
#include <linux/fs.h>
int alloc_chrdev_region(dev_t *first, unsigned int firstminor,
                        unsigned int count, char *name);
```

### ■ Descripción:

- Reserva rango (major,minor) consecutivo cuando *major* desconocido

### ■ Parámetros:

- first: Parámetro de retorno. Primer par (major,minor) que el kernel reserva para el *driver*
- firstminor: Menor *minor number* a reservar dentro del rango consecutivo que otorga el kernel
- count: Número de *minor numbers* a reservar para el *driver*
- name: Nombre del driver (cadena de caracteres arbitraria.) Es el valor que aparecerá en `/proc/devices` al cargar el driver

- **Valor de retorno:** 0 en caso de éxito. En caso de fallo, devuelve valor negativo que codifica el error.

## Descripción de la API (II)

`unregister_chrdev_region()`

```
#include <linux/fs.h>
int unregister_chrdev_region(dev_t first, unsigned int count);
```

### ■ Descripción:

- Liberación de rango de pares (major,minor) del *driver*

### ■ Parámetros:

- `first`: Primer par (major,minor) que el *driver* había reservado previamente
- `count`: Número de *minor numbers* consecutivos que el *driver* había reservado

### ■ Valor de retorno: 0 en caso de éxito. En caso de fallo, devuelve valor negativo que codifica el error.

## Estructura cdev

- Para que el driver pueda recibir peticiones de los programas de usuario, debe crear una estructura cdev e inicializarla adecuadamente

### Operaciones sobre struct cdev <linux/cdev.h>

```
struct cdev *cdev_alloc(void);
```

- Crea estructura cdev y retorna un puntero no nulo a la misma en caso de éxito

```
void cdev_init(struct cdev *p, struct file_operations *fops);
```

- Asocia interfaz de operaciones del driver a estructura cdev

```
int cdev_add(struct cdev *p, dev_t first, unsigned count);
```

- Permite que peticiones de programas de usuario sobre el rango de (major,minor) especificado mediante parámetros first y count sean redirigidas al driver que gestiona estructura cdev



## cdev structure

Operaciones sobre struct cdev <linux/cdev.h>

```
void cdev_del(struct cdev *p);
```

- Elimina asociaciones de estructura cdev (rangos de major/minor) y libera memoria asociada a la estructura

## Ejemplo: chardev.c (1/4)

```
#include <linux/kernel.h>

int init_module(void);
void cleanup_module(void);
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);

#define DEVICE_NAME "chardev" /* Dev name as it appears in /proc/devices */
...
dev_t start; /* Starting (major,minor) pair for the driver */
struct cdev* chardev=NULL; /* Cdev structure associated with the driver */

...

static struct file_operations fops = {
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release
};
```

## Ejemplo chardev.c (2/4)

```
/* This function is called when the module is loaded */
int init_module(void) {
    int major;    /* Major number assigned to our device driver */
    int minor;    /* Minor number assigned to the associated character device */
    int ret;

    /* Get available (major,minor) range */
    if ((ret=alloc_chrdev_region(&start, 0, 1,DEVICE_NAME)){
        ... Error handling ...
        return ret;
    }

    /* Create associated cdev */
    if ((chardev=cdev_alloc())==NULL)
        return -ENOMEM;

    cdev_init(chardev,&fops);

    if ((ret=cdev_add(chardev,start,1)){
        ... Error handling ...
        return ret;
    }

    ...
}
```

## Ejemplo: chardev.c (3/4)

```
/* This function is called when the module is loaded */
int init_module(void) {

    ....

    major=MAJOR(start);
    minor=MINOR(start);

    printk(KERN_INFO "I was assigned major number %d. To talk to\n", major);
    printk(KERN_INFO "the driver, create a dev file with\n");
    printk(KERN_INFO "'sudo mknod -m 666 /dev/%s c %d %d'.\n", DEVICE_NAME, major, minor);
    printk(KERN_INFO "Try to cat and echo to the device file.\n");
    printk(KERN_INFO "Remove the device file and module when done.\n");

    return SUCCESS;
}
```

## Ejemplo: chardev.c (4/4)

```
/* This function is called when the module is unloaded */  
void cleanup_module(void)  
{  
    /* Destroy chardev */  
    if (chardev)  
        cdev_del(chardev);  
  
    /*  
     * Unregister the device  
     */  
    unregister_chrdev_region(start, 1);  
}
```

## operaciones read()/write()

```
static ssize_t device_read(struct file *file, char *buff, size_t len, loff_t *off);  
static ssize_t device_write(struct file *file, const char *buff, size_t len, loff_t *off);
```

### Parámetros relevantes

- `buff`: buffer de bytes o caracteres donde el usuario nos pasa los datos (`write()`) o donde debemos devolverle los datos (`read()`)
- `len`
  - Lectura → número máximo de bytes/caracteres que podemos escribir en `buff`
  - Escritura → número máximo de bytes/caracteres que el usuario escribió y se almacenan en `buff`

### Valor de retorno

- Devuelve número de bytes que el kernel lee de `buff` (en `write()`) o escribe en `buff` (`read()`)
  - 0 → fin de fichero en `read()` (no hay nada más que leer)
- $< 0$  → error

## Punteros al espacio de usuario (I)

- **Puntero al espacio de usuario:** puntero pasado como parámetro en llamada al sistema (p.ej., `read()` o `write()`)
  - Las operaciones `read` y `write` que implementa un driver aceptan como parámetro un puntero al buffer del proceso de usuario
- **No debemos confiar en los punteros al espacio de usuario.** Puntero potencialmente corrupto:
  - Puntero NULL
  - Región de memoria a la que el proceso no tiene acceso



## Punteros al espacio de usuario (II)

- Siempre se ha de trabajar con una copia privada de los datos en espacio de kernel
  - Por ejemplo, declarar array `char kbuf[MAX_CHARS]` local a las funciones `read()` y `write()`
  - En `read()`: trabajar sobre `kbuf` + copiar contenido de `kbuf` a buffer de usuario con `copy_to_user()`
  - En `write()`: copiar datos de buffer de usuario a `kbuf` (`copy_from_user()`) + realizar procesamiento sobre `kbuf`

```
<linux/uaccess.h>
```

```
unsigned long copy_from_user(void *to, const void __user *from,  
                             unsigned long n);  
unsigned long copy_to_user(void __user *to, const void *from,  
                             unsigned long n);
```

- Semántica de copia similar a `memcpy()`
- Ambas funciones devuelven el número de bytes que NO pudieron copiarse



# Ejemplo: chardev (Compilación y Carga)

## Terminal

```
osuser@debian:~/A4Files$ ls
Chardev  Hello
osuser@debian:~/A4Files$ cd Chardev/
osuser@debian:~/A4Files/Chardev$ ls
Makefile  chardev.c
osuser@debian:~/A4Files/Chardev$ make
make -C /lib/modules/4.9.111-lin/build M=/home/osuser/A4Files/Chardev modules
make[1]: Entering directory `/usr/src/linux-headers-4.9.111-lin'
  CC [M]  /home/osuser/A4Files/Chardev/chardev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/osuser/A4Files/Chardev/chardev.mod.o
  LD [M]  /home/osuser/A4Files/Chardev/chardev.ko
make[1]: Leaving directory `/usr/src/linux-headers-4.9.111-lin'
osuser@debian:~/A4Files/Chardev$ sudo insmod chardev.ko
[sudo] password for osuser:
osuser@debian:~/A4Files/Chardev$ lsmod | head
Module                Size  Used by
chardev               2208   0
binfmt_misc           6160   1
uinput                6879   1
nfsd                  198017  2
auth_rpcgss           37914  1 nfsd
oid_registry          2051  1 auth_rpcgss
```

## Ejemplo: chardev (Listado drivers)

### Terminal

```
osuser@debian:~/A4Files/Chardev$ cat /proc/devices
```

```
Character devices:
```

```
1 mem
4 /dev/vc/0
4 tty
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
7 vcs
10 misc
13 input
21 sg
29 fb
108 ppp
128 ptm
136 pts
180 usb
189 usb_device
226 drm
248 chardev
249 hidraw
250 bsg
251 watchdog
```

# Invocar funciones del driver

## Para usar las funciones del driver...

- 1 Crear un fichero especial de caracteres con `mknod` (como root)
  - `sudo mknod -m 666 <ruta_fich_especial> c <major> <minor>`
    - `-m 666`: dar permiso de lectura/escritura a todos los usuarios
  - El *major number* para el driver se puede consultar en el fichero especial `/proc/devices`
- 2 Acceder al fichero de dispositivo creado:
  - Desde un programa de usuario: `open()`, `read()`, `write()`, `close()`
  - Desde el shell: `cat`, `echo`
    - `cat <ruta_fich_especial>` → Lee del fich. de dispositivo hasta EOF (potencialmente varios `read()`) y muestra contenido recibido por pantalla
    - `echo "hola" > <ruta_fich_especial>` → Escribe la cadena "hola\n" (sin el '\0' al final) en el fichero de dispositivo

## Ejemplo: chardev (Creación del fich. especial)

### Terminal

```
osuser@debian:~/A4Files/Chardev$ sudo dmesg | tail
....
[13165.925127] I was assigned major number 248. To talk to
[13165.925130] the driver, create a dev file with
[13165.925132] 'sudo mknod -m 666 /dev/chardev c 248 0'.
[13165.925133] Try to cat and echo to the device file.
[13165.925134] Remove the device file and module when done.
osuser@debian:~/A4Files/Chardev$ sudo mknod -m 666 /dev/chardev c 248 0
osuser@debian:~/A4Files/Chardev$ stat /dev/chardev
  File: `/dev/chardev'
  Size: 0          Blocks: 0          IO Block: 4096   character special file
Device: 4h/4d  Inode: 32859        Links: 1       Device type: f8,0
Access: (0666/crw-rw-rw-)  Uid: (    0/   root)   Gid: (    0/   root)
Access: 2018-12-2 19:49:25.720129709 +0100
Modify: 2018-12-2 19:49:25.720129709 +0100
Change: 2018-12-2 19:49:25.720129709 +0100
 Birth: -
osuser@debian:~/A4Files/Chardev$ ls -l /dev/chardev
crw-rw-rw- 1 root root 248, 0 Dec 2 19:49 /dev/chardev
osuser@debian:~/A4Files/Chardev$
```

## Ejemplo: chardev (Manipulación del disp.)

### Terminal

```
osuser@debian:~/A4Files/Chardev$ cat /dev/chardev
I already told you 0 times Hello world!
osuser@debian:~/A4Files/Chardev$ cat /dev/chardev
I already told you 1 times Hello world!
osuser@debian:~/A4Files/Chardev$ cat /dev/chardev
I already told you 2 times Hello world!
osuser@debian:~/A4Files/Chardev$ echo hello > /dev/chardev
-bash: echo: write error: Operation not permitted
osuser@debian:~/A4Files/Chardev$
```

# Contenido

## 1 Introducción

- Drivers

## 2 Gestión de E/S en Linux

## 3 Módulos del kernel Linux

## 4 Drivers de dispositivos de caracteres

## 5 Práctica 4

# Práctica

## Requisitos

### 1 Es obligatorio usar la máquina virtual de Debian o Ubuntu

- Versión específica del kernel Linux (4.9.x-4.15.x)
- Acceso como administrador (*root*)
  - Ejecutar comando como root vía sudo + password “usuario”:

```
$ sudo <comando>
```

- Iniciar shell de root vía sudo:

```
$ sudo -i
```

### 2 Necesario usar un equipo con teclado estándar para hacer la práctica

- En portátiles sin LEDs (p.ej., Mac) es posible conectar un teclado USB al portátil y hacer que la máquina virtual VMware lo gestione (solo MV Debian)
  - Connecting a second mouse or keyboard directly to a hosted VM