



Gestión de E/S

Sistemas Operativos
Módulo 4



Índice

1. Introducción
2. Hardware de E/S
3. Principios de E/S
4. Software de E/S
5. E/S en Linux
6. Servicios POSIX para la gestión de E/S

Bibliografía:

Sistemas Operativos, J. Carretero [et al.]

Modern Operating Systems, Andrew S. Tanenbaum

P.J. Salzman [et al.], The Linux Kernel Module Programming Guide. 2007-05-18 ver 2.6.4.

<http://www.tldp.org/guides.html> sección kernel version 2.6

Kernel Documentation <http://www.kernel.org/doc/Documentation/>

Índice



- 1. Introducción**
2. Hardware de E/S
3. Principios de E/S
4. Software de E/S
5. E/S en Linux
6. Servicios POSIX para la gestión de E/S



Introducción

- ▶ El corazón de una computadora lo constituye la UCP, 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, ...*



*Dispositivos de entrada
(teclado, ratón)*



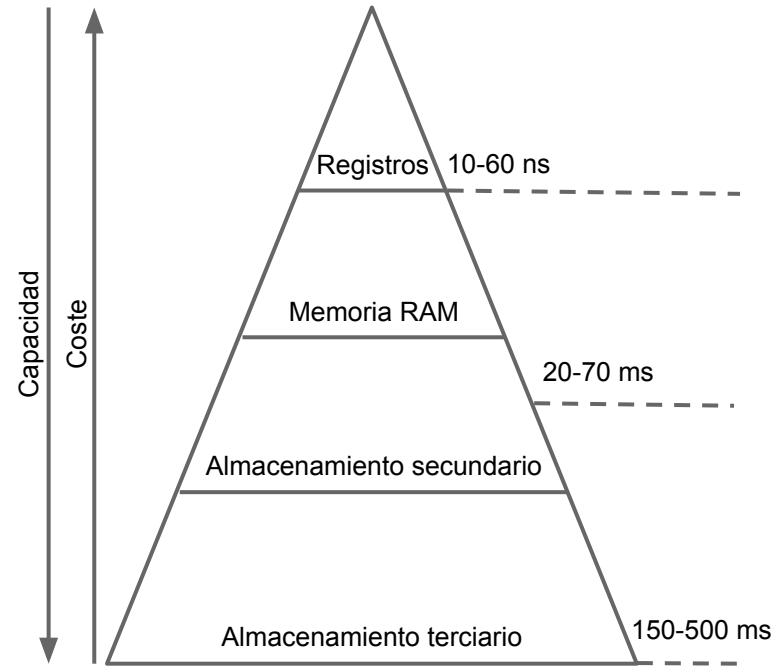
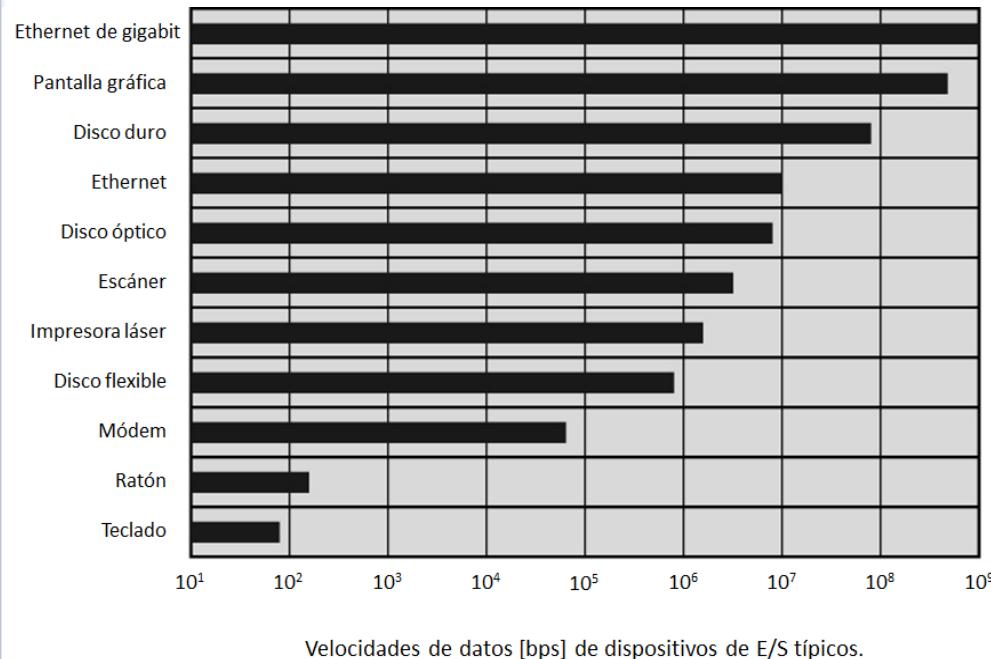
*Unidad principal (UCP, registros,
memoria RAM, Entrada/Salida (discos
internos, red,...))*

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

Velocidad de los dispositivos



- ▶ El gran problema de todos estos dispositivos de E/S es que son muy lentos:
 - ▷ La UCP procesa instrucciones a >1 GHz (<1 ns/ciclo) y la memoria RAM tiene un tiempo de acceso de nanosegundos (10^{-9})
 - ▷ Los dispositivos de E/S más rápidos tienen una velocidad de acceso del orden de milisegundos (10^{-3})
 - ▷ ...





Visión del sistema de E/S

- ▶ La visión del sistema de E/S puede ser muy distinta dependiendo del nivel de detalle necesario en su estudio
 - ▷ Para los programadores: el sistema de E/S es una caja negra que lee y escribe datos en dispositivos externos a través de una funcionalidad bien definida
 - ▷ Para los fabricantes de dispositivos: un dispositivo es un instrumento muy complejo que incluye cientos o miles de componentes electrónicos o electro-mecánicos.
- ▶ Los diseñadores de **sistemas operativos y drivers** se encuentran en un **lugar intermedio** entre los dos anteriores:
 - ▷ Les interesa la funcionalidad del dispositivo, aunque a un nivel de detalle mucho mayor que el requiere el programador de apps.
 - ▶ Necesitan información sobre su comportamiento interno para poder optimizar los métodos de acceso a los mismos
 - ▷ Requieren conocer la arquitectura del software de E/S del SO para poder exponer cada dispositivo al usuario

Funciones del sistema de E/S



- ▶ Facilitar el manejo de los dispositivos periféricos. Para ello debe ofrecer una interfaz entre los dispositivos y el resto del sistema que sea sencilla y fácil de utilizar
- ▶ Optimizar la E/S del sistema, proporcionando mecanismos de incremento de prestaciones donde sea necesario
- ▶ Proporcionar dispositivos virtuales que permitan conectar cualquier tipo de dispositivo físico sin que sea necesario remodelar el sistema de E/S del sistema operativo
- ▶ Permitir la conexión de dispositivos nuevos de E/S, solventando de forma automática su instalación usando mecanismos del tipo plug&play



Índice



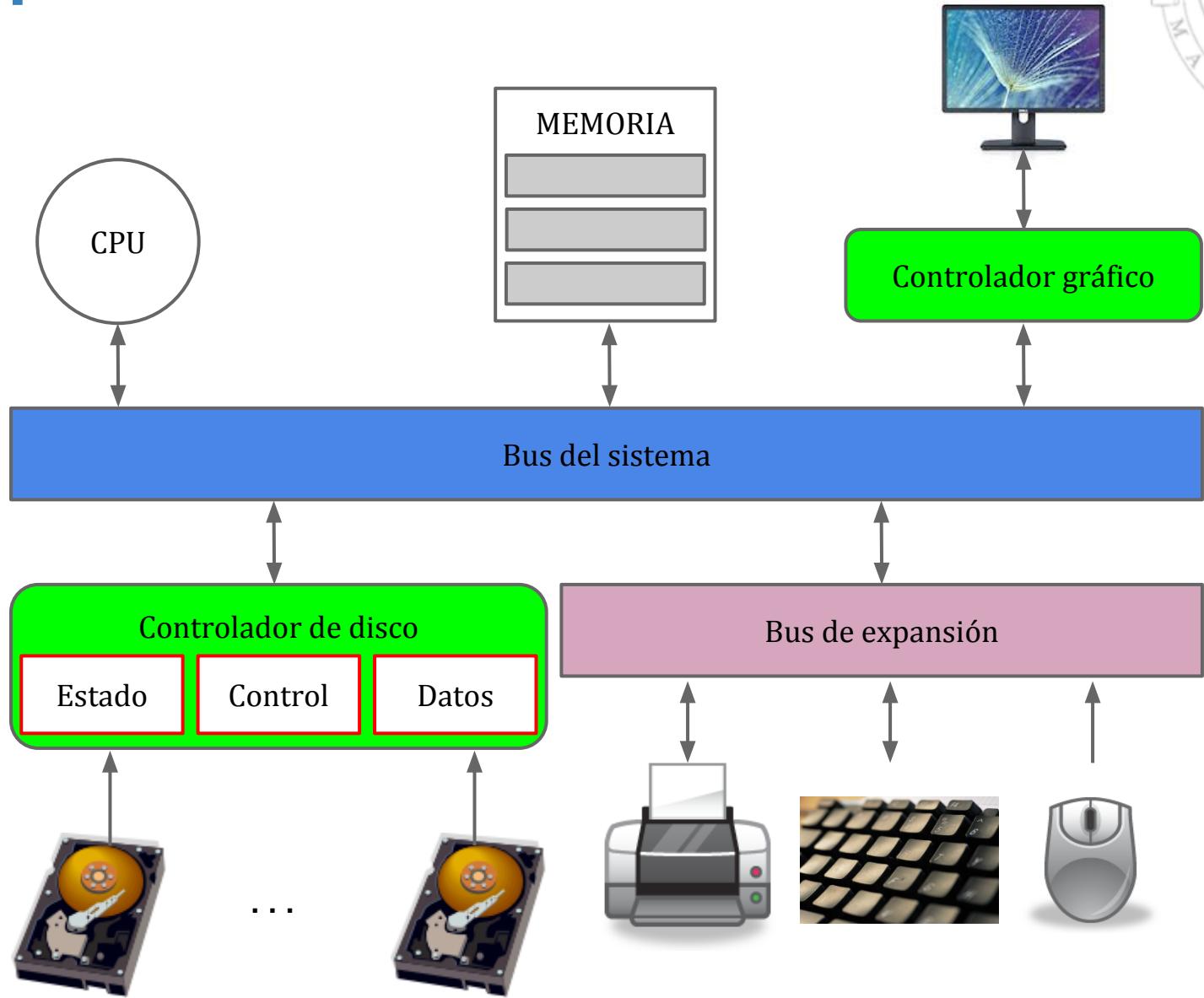
1. Introducción
- 2. Hardware de E/S**
3. Principios de E/S
4. Software de E/S
5. E/S en Linux
6. Servicios POSIX para la gestión de E/S



Conexión de dispositivos de E/S

- ▶ En el modelo de un periférico se distinguen dos elementos:
 - ▷ Periféricos o dispositivos de E/S: elementos que se conectan a la unidad central de proceso a través de controladores hardware
 - ▶ Son el componente mecánico que se conecta al computador.
 - ▷ Controladores 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 (generalmente mediante cables internos o externos).
 - ▶ **Advertencia:**
 - ▷ Controlador HW o unidad de E/S <> Controlador SW o Driver

Conexión de dispositivos a una computadora





Controladores (1/2)

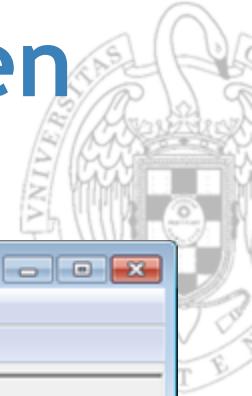
- ▶ Los controladores son muy variados, casi tanto como los dispositivos de E/S
 - ▷ Muchos de ellos, como los de disco, pueden controlar múltiples dispositivos
 - ▷ En los últimos años ha existido un esfuerzo importante de estandarización de los dispositivos, lo que permite usar un mismo controlador para dispositivos de distintos fabricantes
 - ▶ SCSI: Small Computer System Interface
 - ▶ SATA: Serial Advanced Technology Attachment
 - ▶ IDE: Integrated Drive Electronics



Controladores (2/2)

- ▶ El controlador constituye la interfaz del dispositivo con el bus de la computadora
 - ▷ Su programación se lleva a cabo mediante una interfaz de muy bajo nivel que proporciona acceso a una serie de registros del controlador.
 - ▷ **Fundamental:** Dirección de E/S, Unidad de transferencia e Interacción computadora-controlador.
- ▶ Para realizar una operación de E/S, la UCP tiene que escribir sobre los registros del controlador HW
 - ▷ Para ello el SO debe conocer
 - ▶ Función de cada registro (datos o control)
 - ▶ Organización a nivel de bit de cada registro
 - ▶ Dirección de E/S de cada registro
- ▶ **Recuerda:**
 - ▷ E/S aislada
 - ▷ E/S localizada en memoria

Direcciones de “puertos de E/S” en Windows



Recuerda: DMAs

The screenshot shows the Windows Device Manager interface. On the left, the navigation pane includes 'Administración del equipo (local)', 'Herramientas del sistema', 'Programador de tareas', 'Visor de eventos', 'Carpetas compartidas', 'Usuarios y grupos locales', 'Rendimiento', 'Administrador de dispositivos', 'Almacenamiento', 'Administración de discos', and 'Servicios y Aplicaciones'. The main pane displays device details for a virtual machine named 'vbox-jlrisco'. A red box highlights the '4 Controladora de acceso directo a memoria' entry under the 'Acceso directo a memoria (DMA)' heading. The right pane shows 'Acciones' (Actions) and 'Administrador de dispositivos...' (Device Manager). A pink callout bubble points to the highlighted DMA entry with the text 'Recuerda: DMAs'.

Administración de equipos

Archivo Acción Ver Ayuda

vbox-jlrisco

- Acceso directo a memoria (DMA)
 - 4 Controladora de acceso directo a memoria
- Entrada/salida (E/S)
 - [00000000 - 00000CF7] Bus PCI
 - [00000000 - 0000000F] Controladora de acceso directo a memoria
 - [00000020 - 00000021] Controladora programable de interrupciones
 - [00000040 - 00000043] Cronómetro del sistema
 - [00000050 - 00000053] Cronómetro del sistema
 - [00000060 - 00000060] Teclado PS/2 estándar
 - [00000064 - 00000064] Teclado PS/2 estándar
 - [00000080 - 0000008F] Controladora de acceso directo a memoria
 - [000000A0 - 000000A1] Controladora programable de interrupciones
 - [000000C0 - 000000DF] Controladora de acceso directo a memoria
 - [00000170 - 00000177] ATA Channel 1
 - [000001F0 - 000001F7] ATA Channel 0
 - [00000376 - 00000376] ATA Channel 1
 - [00000378 - 0000037F] Puerto de impresora (LPT1)
 - [000003B0 - 000003BB] VirtualBox Graphics Adapter
 - [000003C0 - 000003DF] VirtualBox Graphics Adapter
 - [000003F6 - 000003F6] ATA Channel 0
 - [00000778 - 0000077F] Puerto de impresora (LPT1)
 - [00000D00 - 00000FFFF] Bus PCI
 - [00000D000 - 00000D00F] Controladora IDE principal de bus PCI Intel(R) 82371AB/EB
 - [00000D010 - 00000D017] Adaptador de escritorio Intel(R) PRO/1000 MT
 - [00000D020 - 00000D03F] VirtualBox Device
 - [00000D100 - 00000D1FF] Intel(r) 82801AA AC'97 Audio Controller
 - [00000D200 - 00000D23F] Intel(r) 82801AA AC'97 Audio Controller
 - [00000D240 - 00000D247] Controladora ATA de serie AHCI1.0 estándar
 - [00000D248 - 00000D24B] Controladora ATA de serie AHCI1.0 estándar
 - [00000D250 - 00000D257] Controladora ATA de serie AHCI1.0 estándar
 - [00000D258 - 00000D25B] Controladora ATA de serie AHCI1.0 estándar
 - [00000D260 - 00000D26F] Controladora ATA de serie AHCI1.0 estándar

Direcciones de puertos de E/S en Linux



```
Terminal
| usuario@debian:~$ cat /proc/ioports
 0000-0ca1 : PCI Bus 0000:00
 0000-001f : dma1
 0020-0021 : pic1
 0040-0043 : timer0
 0050-0053 : timer1
 0060-0060 : keyboard
 0064-0064 : keyboard
 0070-0071 : rtc0
 0080-008f : dma page reg
 00a0-00a1 : pic2
 00c0-00df : dma2
 00f0-00ff : fpu
 0170-0177 : 0000:00:1f.1
    0170-0177 : piix
 01f0-01f7 : 0000:00:1f.1
    01f0-01f7 : piix
 02f8-02ff : serial
 0300-031f : 0000:00:1f.3
    0300-031f : i801_smbus
 0376-0376 : 0000:00:1f.1
    0376-0376 : piix
 03c0-03df : vga+
 ...
...
```



Índice



1. Introducción
2. Hardware de E/S
- 3. Principios de E/S**
4. Software de E/S
5. E/S en Linux
6. Servicios POSIX para la gestión de E/S

Principios de E/S



- ▶ **Recuerda:**
 - ▷ E/S programada
 - ▷ E/S por interrupciones
 - ▷ E/S con DMA
- ▶ **Interrupción:** señal que un dispositivo de E/S envía al procesador cuando requiere su atención
 - ▷ **Motivación interrupciones:**
 - ▶ Diferencia de velocidad entre CPU y dispositivos de E/S
 - ▶ Evitar polling
 - ▷ Al recibir una interrupción el procesador ejecuta una Rutina de Tratamiento de Interrupción (RTI) bajo control del SO
 - ▶ Procesamiento típico:
 - ▷ Notificar al HW del tratamiento de la interrupción
 - ▷ Transferir datos entre dispositivo y memoria principal
 - ▷ Reseteo de HW para siguiente interrupción
 - ▷ ...



Índice



1. Introducción
2. Hardware de E/S
3. Principios de E/S
- 4. Software de E/S**
5. E/S en Linux
6. Servicios POSIX para la gestión de E/S



Drivers (1/4)

- ▶ Al software en el SO destinado a gestionar E/S con un dispositivo se le llama **driver o manejador de dispositivo (device driver)**
- ▶ Un dispositivo puede **generar** y/o **recibir** datos
- ▶ Algunos dispositivos pueden necesitar un **control** específico como por ejemplo, **rebobinar**
- ▶ ⇒ Un diseño de un **driver genérico** de SO debería incluir dos clases de funciones
 - ▷ open(), read(), write(), close()
 - ▷ control(), handle_event()
- ▶ ¡Los SSOO tipo UNIX ofrecen una interfaz similar para dispositivos, ficheros, ..., hasta dispositivos de red!



Drivers (2/4)

- ▶ Un driver abstrae un dispositivo hardware específico en un modelo **genérico** de E/S del dispositivo
- ▶ El resultado es un conjunto de modelos abstractos de discos, cintas, pantallas, etc.
- ▶ Simplifica la labor de portar SSOO y aplicaciones a nuevas plataformas hardware - **independencia del dispositivo**
- ▶ Internamente, un driver está programado con ciertos detalles del **dispositivo físico**, pero presenta **independencia total en la interfaz**, es decir, en el protocolo de llamadas del driver, que es común a todos los dispositivos



Drivers (3/4)

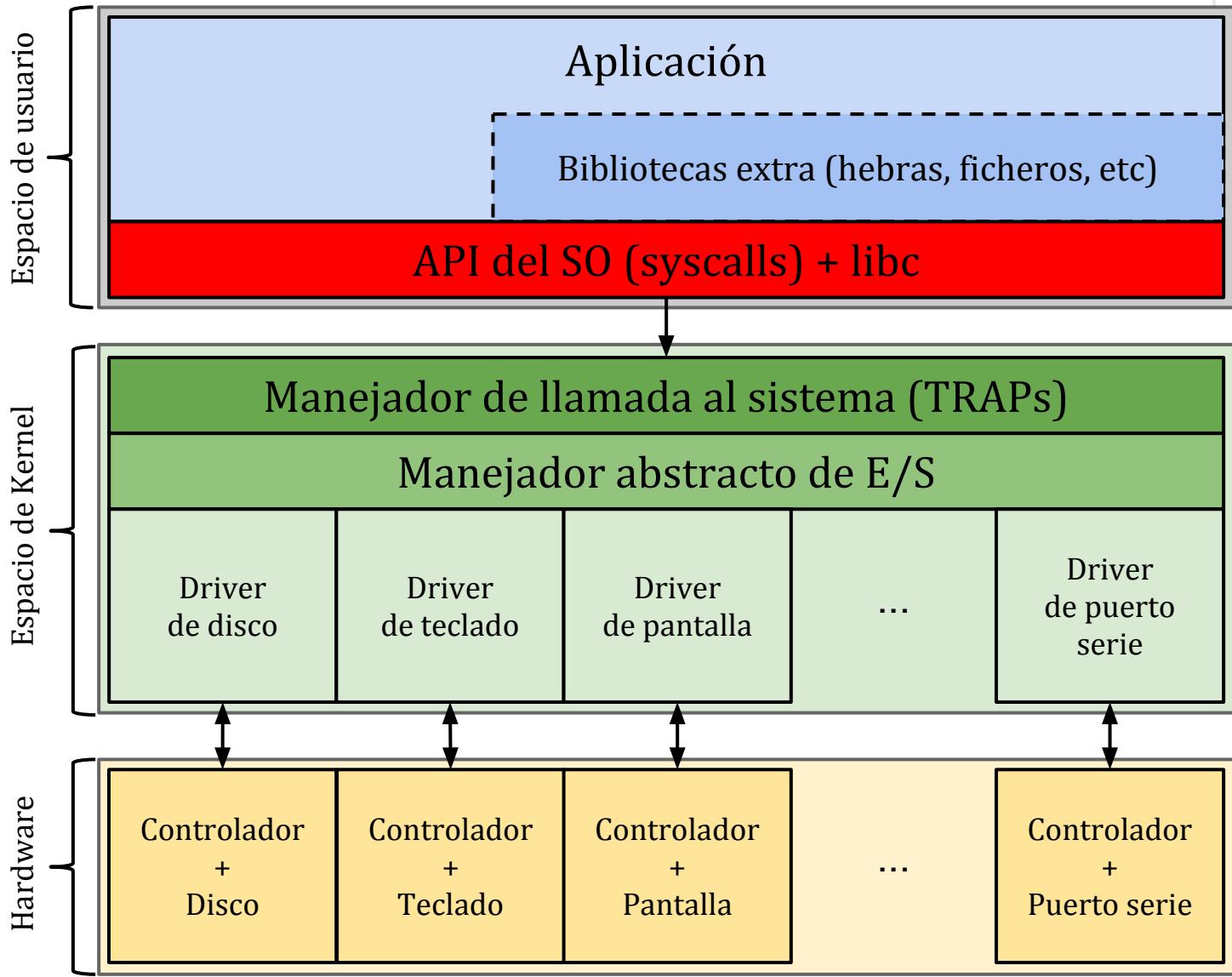
- ▶ Sin embargo, no podemos ser tan, tan genéricos
- ▶ Al final, los dispositivos se tienen que dividir en un pequeño número de clases:
 - ▷ Dispositivos de carácter
 - ▶ puerto serie, teclado, ratón, ...
 - ▷ Dispositivos de bloque
 - ▶ disco, red, pantalla, ...
- ▶ Estas clases o categorías se deben incluir para definir diferentes protocolos en la interfaz abstracta o genérica del driver y así ganar en rendimiento de la E/S



Drivers (4/4)

- ▶ Los drivers tienen que lidiar con las siguientes características de los dispositivos
 - ▷ E/S por encuesta, interrupciones o DMA
 - ▷ Mapeo en memoria (E/S ailada, centralizada en memoria)
 - ▷ Comandos específicos del controlador del dispositivo, etc.
- ▶ Por lo que, un driver estándar contiene las siguientes funciones
 - ▷ inicializar el dispositivo
 - ▷ comienzo de operación de escritura
 - ▷ comienzo de operación de lectura
 - ▷ control del dispositivo
 - ▷ apagar el dispositivo
 - ▷ rutinas de tratamiento de interrupción (RTIs)

Visión general



Ejemplo de escritura a disco (1/2)



- ▶ Parte “productor” del driver:
 - ▷ El usuario realiza una **llamada al sistema** para escribir unos bloques de datos en disco
 - ▷ El procesador cambia de modo usuario a **modo kernel** (TRAP)
 - ▷ Se invoca al **manejador de llamadas a sistema** y se comprueban los parámetros de la llamada
 - ▷ Se llama al **manejador abstracto de E/S**
 - ▷ Se copian los **datos de usuario** a un **buffer del SO** (o se salvan (locked), para prevenir sobreescrituras durante la E/S)
 - ▷ El driver añade el **buffer a la cola de E/S pendientes**
 - ▶ Esto varía en función del tipo de dispositivo (bloque o car.)
 - ▷ Se vuelve al **manejador genérico** que **suspende el proceso** en E/S y continua con otro proceso listo

Ejemplo de escritura a disco (2/2)



- ▶ Parte “consumidor” del driver:
 - ▷ Cuando tiene lugar una **interrupción de fin de transferencia de disco**, la RTI correspondiente añade información del resultado de la transferencia al buffer y borra el estado de bloqueo del proceso en E/S (para el Planificador del SO)
 - ▷ Se mira en la cola Q de **transferencias pendientes** y se configuran los registros del DMA y comienza la nueva transferencia
 - ▷ Se vuelve de la RTI
 - ▷ **Notas**
 - ▶ Si no hay E/S pendiente la RTI sólo vuelve
 - ▶ Si la cola Q está vacía cuando se llama a start, comienza la transferencia de inmediato

Dificultades de la gestión de interrupciones en el SO



- ▶ Formato de RTI y gestión de interrupciones es específico de plataforma
 - ▷ Complica el desarrollo/mantenimiento de drivers para distintas arquitecturas y controladores de interrupciones
 - ▷ Solución: API independiente de plataforma + Notificaciones al driver
 - ▶ El SO instala la RTI en el proceso de arranque
 - ▶ Al generarse la interrupción la RTI invoca una función del driver que realiza el procesamiento asociado (manejador de interrupción)





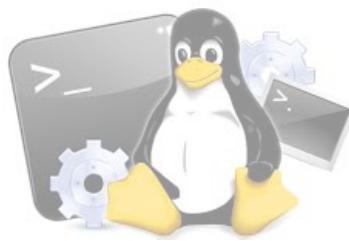
Índice

1. Introducción
2. Hardware de E/S
3. Principios de E/S
4. Software de E/S
- 5. E/S en Linux**
6. Servicios POSIX para la gestión de E/S

Ficheros de dispositivo



- ▶ Casi todos los dispositivos de E/S se representan como ficheros especiales (que pueden ser de **bloque** o **carácter**)
 - ▷ /dev/hda1 para la primera partición del primer disco IDE
 - ▷ /dev/lp0 para la impresora
- ▶ Para cada fichero especial hay **asociado** un driver de dispositivo
- ▶ Cada **driver** tiene un **número de dispositivo principal (major)** que sirve para identificarlo
 - ▷ Si el driver sirve a varios dispositivos, cada **dispositivo** tiene un **número de dispositivo secundario (minor)** que lo identifica
- ▶ El par (major, minor) identifica de forma única cada dispositivo de E/S



Ficheros de dispositivo



- ▶ Los ficheros de dispositivo se alojan *por convenio* en el directorio `/dev`.
- ▶ Para ver los ficheros de dispositivo presentes en el sistema basta con listar el contenido del directorio `/dev`

The screenshot shows a terminal window titled "Terminal" with the command `$ ls -l /dev` entered. A green callout bubble points to the word "Major" in the output, which is part of the device node names. The output lists various device nodes with their major and minor numbers, file types (e.g., brw-rw----, crw-rw----), owners (root), groups (disk, uucp, audio, root), creation dates (Nov 19, Dec 2), and times (10:20). The output ends with "random".

```
$ ls -l /dev
...
brw-r----- 1 root disk 3, 0 Nov 19 10:20 hda
brw-r----- 1 root disk 3, 1 Nov 19 10:20 hda1
brw-r----- 1 root disk 3, 2 Nov 19 10:20 hda2
...
crw-rw---- 1 root uucp 4, 64 Nov 19 10:20 ttys0
crw-rw---- 1 root uucp 4, 65 Nov 19 10:20 ttys1
...
crw-rw---- 1 root audio 14, 3 Dec 2 00:31 dsp
crw-rw---- 1 root audio 14, 4 Dec 2 00:31 audio
...
crw-rw-rw- 1 root root 1, 8 Nov 19 10:20 random
```



Ficheros de dispositivo

- ▶ Los ficheros de dispositivo son un potente mecanismo de trabajo con varios dispositivos hardware tal y como si fuesen ficheros ordinarios.

```
# dd if=/dev/hda of=mbr.bin bs=512 count=1
```

- ▷ **Descripción:** El comando leerá los primeros 512 bytes desde el comienzo del disco duro (el Master Boot Record, MBR) y lo almacenará en el archivo mbr.bin (dd es un comando que copia parte de un fichero en otro).

```
# dd if=/dev/zero of=/dev/hda
```

- ▷ **Descripción:** Escribe ceros en todo el disco duro, eliminando toda la información existente
- ▶ ¿Cómo sabe el sistema operativo a qué dispositivo está asociado un fichero de dispositivo? ⇒ (major, minor)



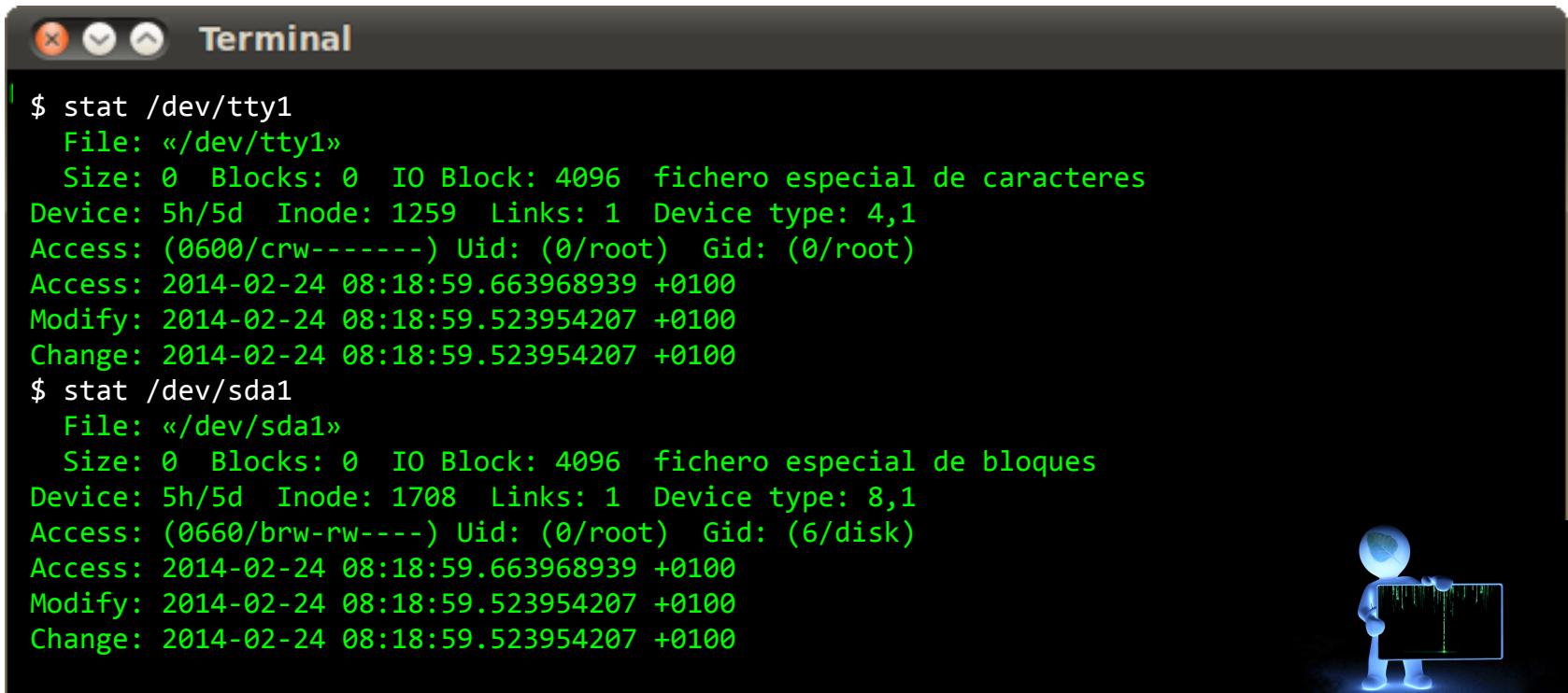
Ficheros de dispositivo

- ▶ Se puede crear un nuevo fichero de dispositivo usando el comando mknod
 - # `mknod /dev/<nombre> <tipo> <núm_major> <núm_minor>`
- ▶ donde:
 - ▷ `<nombre>`: Nombre de archivo de dispositivo
 - ▷ `<tipo>`: **c** para dispositivos tipo carácter y **b** para dispositivos tipo bloque
 - ▷ `<núm_major>` y `<núm_minor>`: major y minor del driver del dispositivo al que este fichero queda asociado
- ▶ Se pueden crear ficheros de dispositivo con cualquier major y minor, sólamente útil si existe un driver asociado con los mismos números.



Major and minor number

- ▶ El comando `stat` permite consultar el tipo de dispositivo asociado al fichero así como el major y minor number del mismo



A screenshot of a terminal window titled "Terminal". The window shows the output of the `stat` command run twice: once on `/dev/tty1` and once on `/dev/sda1`. The output is displayed in green text on a black background. The terminal has a standard window title bar with close, minimize, and maximize buttons.

```
$ stat /dev/tty1
  File: «/dev/tty1»
  Size: 0  Blocks: 0  IO Block: 4096  fichero especial de caracteres
Device: 5h/5d  Inode: 1259  Links: 1  Device type: 4,1
Access: (0600/crw-----) Uid: (0/root)  Gid: (0/root)
Access: 2014-02-24 08:18:59.663968939 +0100
Modify: 2014-02-24 08:18:59.523954207 +0100
Change: 2014-02-24 08:18:59.523954207 +0100
$ stat /dev/sda1
  File: «/dev/sda1»
  Size: 0  Blocks: 0  IO Block: 4096  fichero especial de bloques
Device: 5h/5d  Inode: 1708  Links: 1  Device type: 8,1
Access: (0660;brw-rw----) Uid: (0/root)  Gid: (6/disk)
Access: 2014-02-24 08:18:59.663968939 +0100
Modify: 2014-02-24 08:18:59.523954207 +0100
Change: 2014-02-24 08:18:59.523954207 +0100
```



Asociación entre driver y major



```
Terminal
$ cat /proc/devices
Character devices:
 1 mem
 4 /dev/vc/0
 4 tty
 4 ttys
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
 6 lp
...
136 pts
180 usb
189 usb_device
...
Block devices:
 2 fd
259 blkext
 7 loop
 8 sd
11 sr
65 sd
66 sd
67 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 de dispositivo se implementan como **módulos** cargables **del kernel**



Módulos

- ▶ Un driver puede ser estáticamente enlazado “dentro” del kernel o compilado como módulos del kernel.
 - ▷ Entonces, un módulo es una parte del kernel que puede cargarse y descargarse del kernel bajo demanda.
 - ▷ Esto es más conveniente que el enlazado estático, porque de esta manera se puede añadir nueva funcionalidad al kernel cuando se necesite
- ▶ Gestión de módulos en Linux
 - ▷ `lsmod`: lista los módulos cargados actualmente.
 - ▷ `modinfo`: nos da información sobre un módulo.
 - ▷ `insmod`: carga un módulo. Interfaz de bajo nivel.
 - ▷ `rmmod`: descarga un módulo.
 - ▷ `modprobe`: interfaz de alto nivel para cargar módulos.
 - ▶ Busca en `/etc/modprobe` información y ruta de los módulos



Módulos vs. Aplicaciones

Aplicaciones	Módulos
Modo usuario	Modo kernel
Cualquier función de biblioteca disponible	Sólo símbolos exportados por el kernel <ul style="list-style-type: none">• /proc/kallsyms• libc no disponible• printk()
Realizan su función de principio a fin (main)	Ejecutan su función de inicio “init_module” al registrarse y quedan residentes para dar servicio

En los módulos no podemos usar printf (libc). El kernel proporciona una función similar, printk, que permite escribir mensajes en los ficheros de log y por consola virtual (no terminal).

Ejemplo - Implementación (hello.c)



```
File Edit Options Buffers Tools Help
... 
/*
 * hello.c - EL módulo más simple y menos original posible.
 */
#include <linux/module.h> /* Requerido por todos los módulos */
#include <linux/kernel.h> /* Requerido por KERN_INFO */
MODULE_LICENSE("GPL");
int init_module(void) {
    printk(KERN_INFO "Hello world.\n");
    /*
     * Retorno != 0 implica fallo de init_module; el módulo no
     * puede cargarse.
     */
    return 0;
}
void cleanup_module(void) {
    printk(KERN_INFO "Goodbye world.\n");
}
```

The code is a simple Linux kernel module named 'hello.c'. It includes the necessary headers for modules and kernels. It defines two functions: 'init_module' and 'cleanup_module'. The 'init_module' function prints 'Hello world.' to the kernel log using the macro 'printk' (highlighted with a red box). The 'cleanup_module' function prints 'Goodbye world.' to the log when the module is unloaded. The code is annotated with comments explaining its purpose and the meaning of the return value from 'init_module'.

Ver símbolos (variables y funciones) exportados por el kernel:
\$ cat /proc/kallsyms
Ver salida de printk:
cat /var/log/messages

-U:--- ejemplo All L1 (Fundamental)-----



Ejemplo - Compilación (Makefile)

- ▶ Los módulos del kernel deben compilarse de forma diferente a los ficheros C habituales.

The screenshot shows a text editor window with a menu bar (File, Edit, Options, Buffers, Tools, Help) and a toolbar with various icons. The main area contains a Makefile for a kernel module named 'hello'. The code is as follows:

```
MODULE_NAME=hello

obj-m := ${MODULE_NAME}.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 - Test



```
Terminal

$ make
make -C /lib/modules/2.6.32-5-686/build M=/home/usuario modules
make[1]: se ingresa al directorio `/usr/src/linux-headers-2.6.32-5-686'
CC [M]  /home/usuario/hello.o
Building modules, stage 2.
MODPOST 1 modules
CC      /home/usuario/hello.mod.o
LD [M]  /home/usuario/hello.ko
make[1]: se sale del directorio `/usr/src/linux-headers-2.6.32-5-686'
# insmod hello.ko
# lsmod
Module           Size  Used by
hello            528   0
ppdev            4058  0
lp               5570  0
# tail /var/log/messages
Oct 30 22:38:25 VMWare-debian6 vmusr[2524]: [ warning] [Gtk]
gtk_disable_setlocale() must be called before gtk_init()
Oct 30 22:40:32 VMWare-debian6 kernel: [ 176.371767] Hello world.
# rmmod hello
```





Implementar un driver como módulo

- ▶ Cuando el driver es cargado debe registrarse a sí mismo en el kernel especificando el major y las operaciones que el driver soporta
 - ▷ Dispositivos tipo carácter: `register_chrdev()`
 - ▷ Dispositivos tipo bloque: `register_blkdev()`

```
int register_chrdev(unsigned int major,  
                    const char* name,  
                    struct file_operations* fops);
```

major: número major requerido (0 si queremos que se asigne automáticamente)

name: nombre del dispositivo como aparecerá en /proc/devices

fops: puntero a una estructura `file_operations` que indica qué operaciones maneja el driver



La estructura file_operations

```
File Edit Options Buffers Tools Help
[Icons]
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 (*readdir) (struct file *, void *, filldir_t);
    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 (*lock) (struct file *, int, struct file_lock *);
    ssize_t(*readv) (struct file *, const struct iovec *, unsigned long,
                    loff_t *);
    ssize_t(*writev) (struct file *, const struct iovec *, unsigned long,
                     loff_t *);
    ssize_t(*sendfile) (struct file *, loff_t *, size_t, read_actor_t,
                       void __user *);
    ssize_t(*sendpage) (struct file *, struct page *, int, size_t, loff_t *,
                      int);
    // ...
};

-U:--- ejemplo All L1 (Fundamental)-----
http://www.tldp.org/LDP/lkmpg/2.6/html/x569.html
```



La estructura file_operations

- ▶ No todos los campos de esta estructura deben inicializarse, sólamente aquellas que se corresponden con operaciones soportadas por el driver del dispositivo, por ejemplo:

The screenshot shows a code editor window with a menu bar (File, Edit, Options, Buffers, Tools, Help) and a toolbar with various icons. The main area contains the following C code:

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

chardev.c: Dispositivo tipo carácter



```
File Edit Options Buffers Tools Help
... 
/*
 * chardev.c: Creates a read-only char device that says how many times
 * you've read from the dev file
 */
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h> /* for put_user */

/*
 * Prototypes - this would normally go in a .h file
 */
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 SUCCESS 0
#define DEVICE_NAME "chardev" /* Dev name as it appears in /proc/devices */
#define BUF_LEN 80 /* Max length of the message from the device */
// ...

-U:--- ejemplo All L1 (Fundamental)-----
```

chardev.c: Dispositivo tipo carácter



```
File Edit Options Buffers Tools Help
[File Icons]
/*
 * Global variables are declared as static, so are global within the file.
 */
static int Major; /* Major number assigned to our device driver */
static int Device_Open = 0; /* Is device open?
                             * Used to prevent multiple access to device */
static char msg[BUF_LEN]; /* The msg the device will give when asked */
static char *msg_Ptr;
static struct file_operations fops = {
    .read = device_read,
    .write = device_write,
    .open = device_open,
    .release = device_release
};

/* Called when a process writes to dev file: echo "hi" > /dev/hello */
static ssize_t device_write(struct file *filp, const char *buff, size_t len,
loff_t * off) {
    printk(KERN_ALERT "Sorry, this operation isn't supported.\n");
    return -EINVAL;
}

// ...

-U:--- ejemplo      All L1 (Fundamental)-----
```

chardev.c: Dispositivo tipo carácter



```
File Edit Options Buffers Tools Help
... 

```
// This function is called when the module is loaded
int init_module(void) {
 Major = register_chrdev(0, DEVICE_NAME, &fops);
 if (Major < 0) {
 printk(KERN_ALERT "Registering char device failed with %d\n", Major);
 return Major;
 }
 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 "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major);
 printk(KERN_INFO "Try various minor numbers. Try to cat and echo to\n");
 printk(KERN_INFO "the device file.\n");
 printk(KERN_INFO "Remove the device file and module when done.\n");
 return SUCCESS;
}

/*
 * This function is called when the module is unloaded
 */
void cleanup_module(void) {
 unregister_chrdev(Major, DEVICE_NAME); /* Unregister the device */
}
// ...
```



- U: --- ejemplo All L1 (Fundamental) -----


```

chardev.c: Dispositivo tipo carácter



```
File Edit Options Buffers Tools Help
... icons ...

/* Called when a process tries to open the device file, like
 * "cat /dev/mycharfile" */
static int device_open(struct inode *inode, struct file *file) {
    static int counter = 0;
    if (Device_Open)
        return -EBUSY;
    Device_Open++;
    sprintf(msg, "I already told you %d times Hello world!\n", counter++);
    msg_Ptr = msg;
    try_module_get(THIS_MODULE);
    return SUCCESS;
}

/* Called when a process closes the device file. */
static int device_release(struct inode *inode, struct file *file) {
    Device_Open--;
    /* We're now ready for our next caller */
    /* Decrement the usage count, or else once you opened the file, you'll
     * never get rid of the module. */
    module_put(THIS_MODULE);
    return 0;
}
// ...

-U:--- ejemplo Al
```

Previenen el borrado de un módulo cuando está en uso. Incrementa/Decrementa un contador interno del kernel para cada módulo. Si el contador es distinto de cero, el módulo no puede eliminarse. Se pueden ver los valores de estos contadores con lsmod (Columna “Used by”).

chardev.c: Dispositivo tipo carácter



```
File Edit Options Buffers Tools Help
... 


```
/* Called when a process, which already opened the dev file, attempts to
 * read from it. */
static ssize_t device_read(struct file *filp, /* see include/linux/fs.h */
 char *buffer, /* buffer to fill with data */
 size_t length, /* length of the buffer */
 loff_t * offset) {
 /* Number of bytes actually written to the buffer */
 int bytes_read = 0;
 /* If we're at the end of the message, return 0 signifying end of file */
 if (*msg_Ptr == 0)
 return 0;
 /* Actually put the data into the buffer */
 while (length && *msg_Ptr) {
 /* The buffer is in the user data segment, not the kernel segment so
 * "*" assignment won't work. We have to use put_user which copies data
 * from the kernel data segment to the user data segment. */
 put_user(*msg_Ptr++, buffer++);
 length--;
 bytes_read++;
 }
 /* Most read functions return the number of bytes put into the buffer */
 return bytes_read;
}
// ...
```



- U: --- ejemplo


```

La función `put_user` debe usarse cuando se transfieren datos entre los espacios de kernel y de usuario. La dirección de `buffer` no se puede usar directamente por problemas de seguridad. También se puede usar la función `get_user` o `copy_from_user` (consultar las páginas del manual).

chardev.c: Makefile



```
File Edit Options Buffers Tools Help
...  ...
MODULE_NAME=chardev

obj-m := ${MODULE_NAME}.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

=

-U:--- ejemplo All L1 (Fundamental)-----
```

chardev.c: Test



```
Terminal
$ make
...
# insmod chardev.ko
# tail /var/log/messages
Buscar en el archivo de log el mayor asignado a este módulo (251)
# mknod /dev/chardev c 251 0
# cat /dev/chardev
I already told you 0 times Hello world!
# cat /dev/chardev
I already told you 1 times Hello world!
# cat /dev/chardev
I already told you 2 times Hello world!
Si intentamos escribir en el dispositivo obtendremos un mensaje de error, en
el terminal o en el archivo "log":
# echo "Hello" > /dev/chardev
bash: echo: write error: Invalid argument
# rmmod /dev/chardev
¿Por qué aparece este error?
No obstante, enhorabuena, ¡acabamos de crear nuestro primer driver!
```



Índice



1. Introducción
2. Hardware de E/S
3. Principios de E/S
4. Software de E/S
5. E/S en Linux
- 6. Servicios POSIX para la gestión de E/S**



Servicios POSIX para gestión de E/S

- ▶ Servicios de fecha y hora
 - ▷ **time_t time(time_t* t)** ⇒ Obtiene la fecha y hora como el número de segundos transcurridos desde el 1 de enero de 1970 en UTC. Ver también: gmtime, localtime
 - ▷ **int stime(time_t* t)** ⇒ Fija la fecha y hora conforme al parámetro recibido
- ▶ Servicios de temporización
 - ▷ **unsigned int alarm(unsigned int seconds)** ⇒ Permite establecer un temporizador. Tras el plazo dado en segundos, se activa la señal SIGALRM

Servicios POSIX para gestión de E/S



- ▶ Servicios de contabilidad
 - ▷ `clock_t times(struct tms* info)` ⇒ Devuelve información sobre el tiempo de ejecución de un proceso y sus hijos
- ▶ Servicios de E/S sobre dispositivos
 - ▷ `open, read, write, close` (ya vistos en gestión de ficheros)
 - ▷ `int ioctl(int descriptor, int petición, ...)` ⇒ Permite configurar opciones específicas de cada dispositivo o realizar operaciones que no se pueden expresar con read o write (como, por ejemplo, rebobinar una cinta). El primer argumento es el id del descriptor de fichero asociado al dispositivo, el segundo es un identificador de la operación y que depende del dispositivo