Grupo ARCOS
Universidad Carlos III de Madrid

Lección 3c

procesos, periféricos, *drivers* y servicios ampliados

Diseño de Sistemas Operativos
Grado en Ingeniería Informática

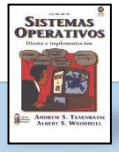
CC BY NC SA

Lecturas recomendadas

Base

1. Carretero 2007:
   1. Cap.7

Recomendada



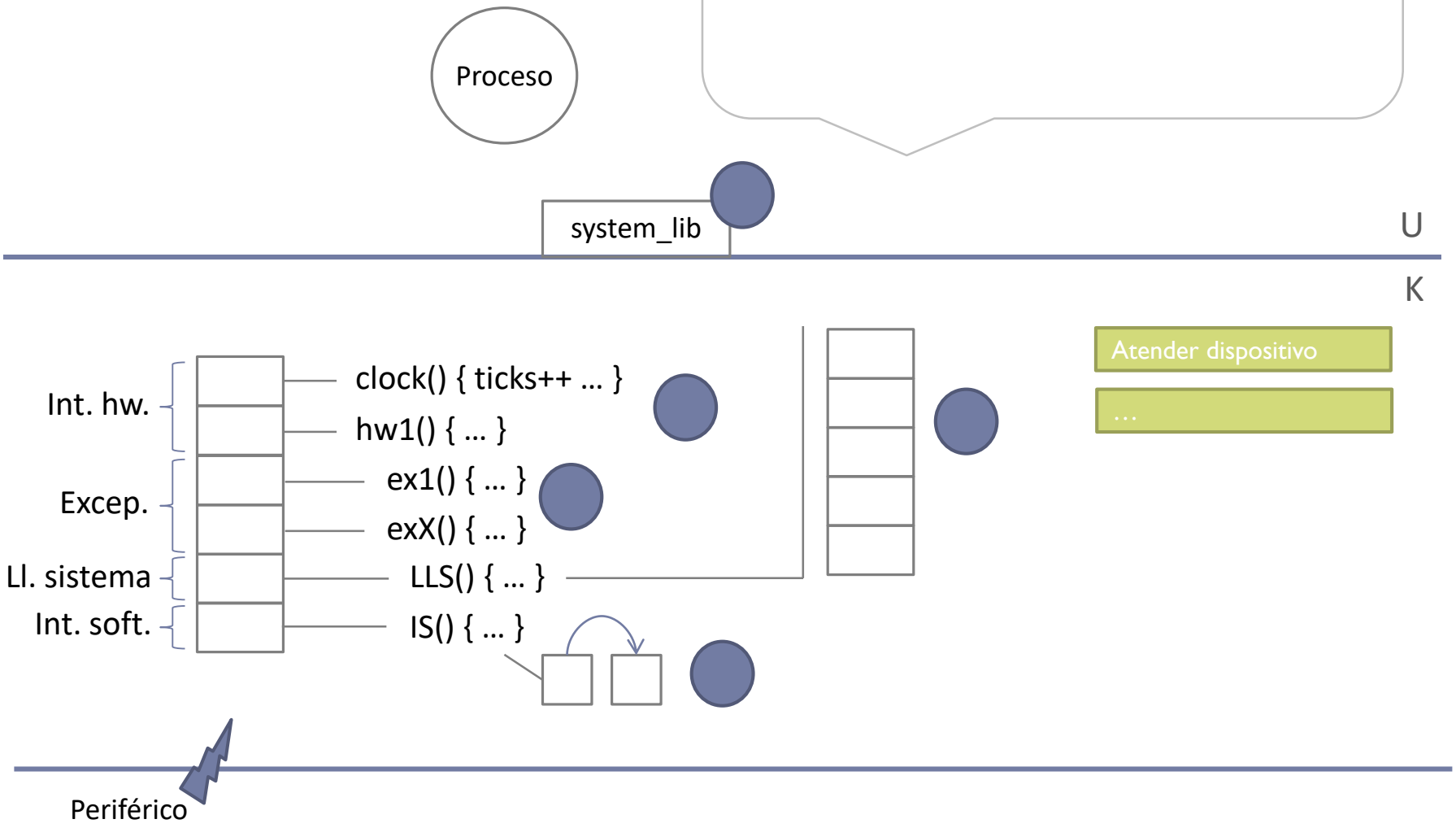
1. Tanenbaum 2006(en):
   1. Cap.3
2. Stallings 2005(en):
   1. Parte tres
3. Silberschatz 2006:
   1. Cap. Sistemas de E/S

A recordar…

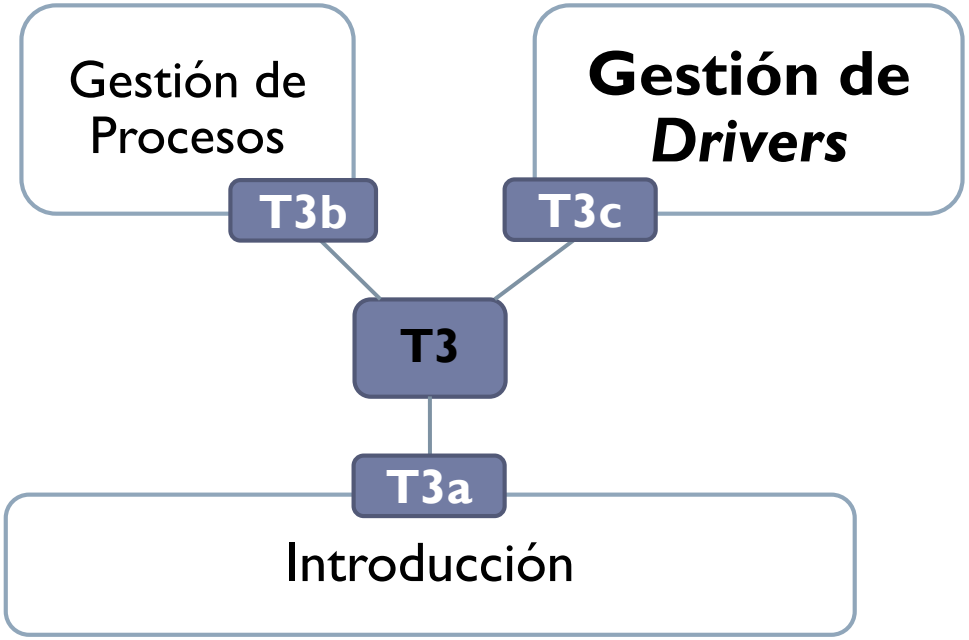
1. Estudiar la teoría asociada.
   - Estudiar el material asociado a la bibliografía: las transparencias solo no son suficiente.
2. Repasar lo visto en clase.
   - Realizar el cuaderno de prácticas progresivamente.
3. Ejercitar las competencias.
   - Realizar las prácticas progresivamente.
   - Realizar todos los ejercicios posibles.

Contexto…

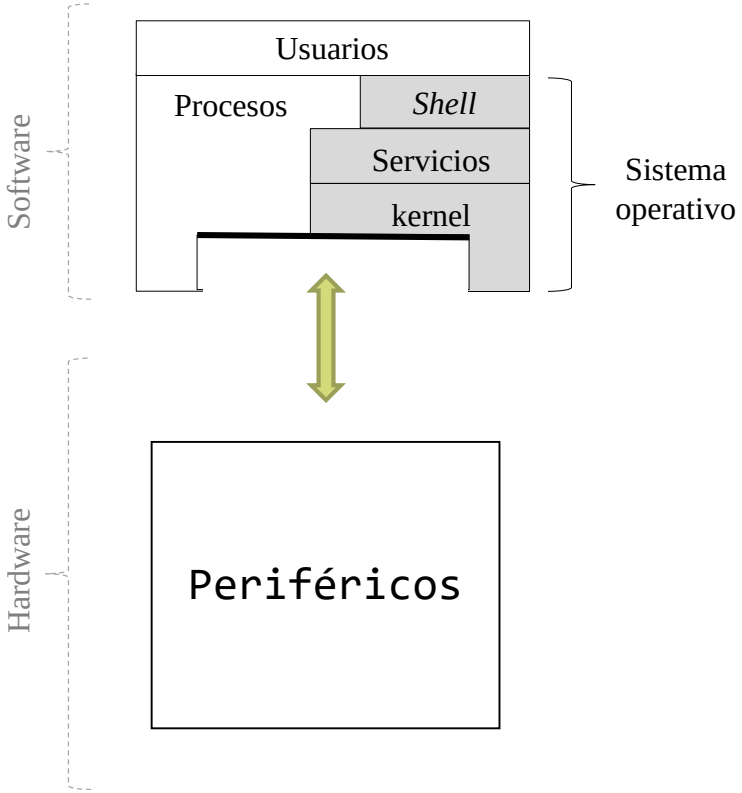
En el tema 3 se introducirá los aspectos del funcionamiento relativos a la **gestión de dispositivos**…



Contexto…

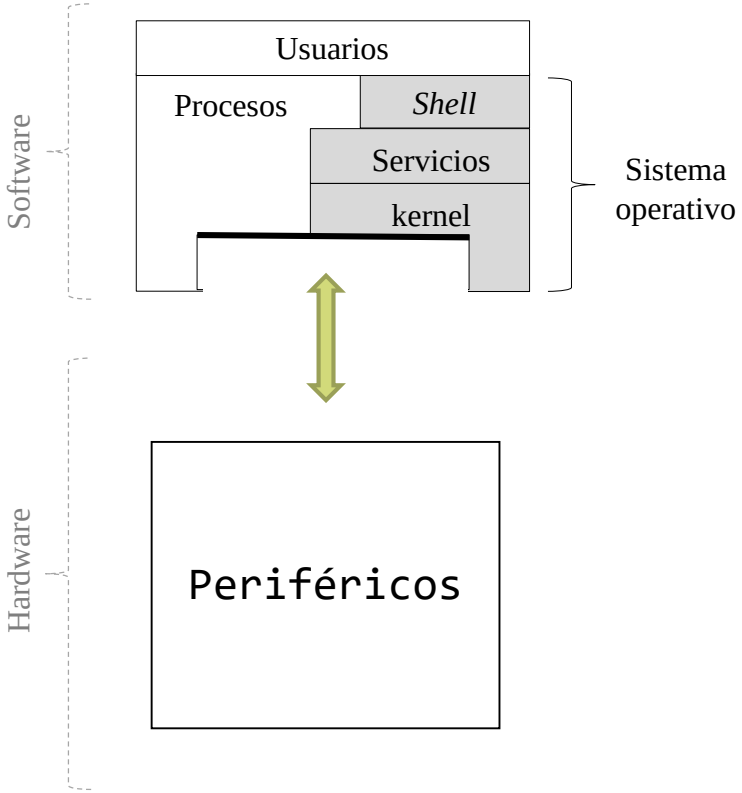

Gestión de Procesos
Gestión de *Drivers*
T3b
T3c
T3
T3a
Introducción

Contenidos



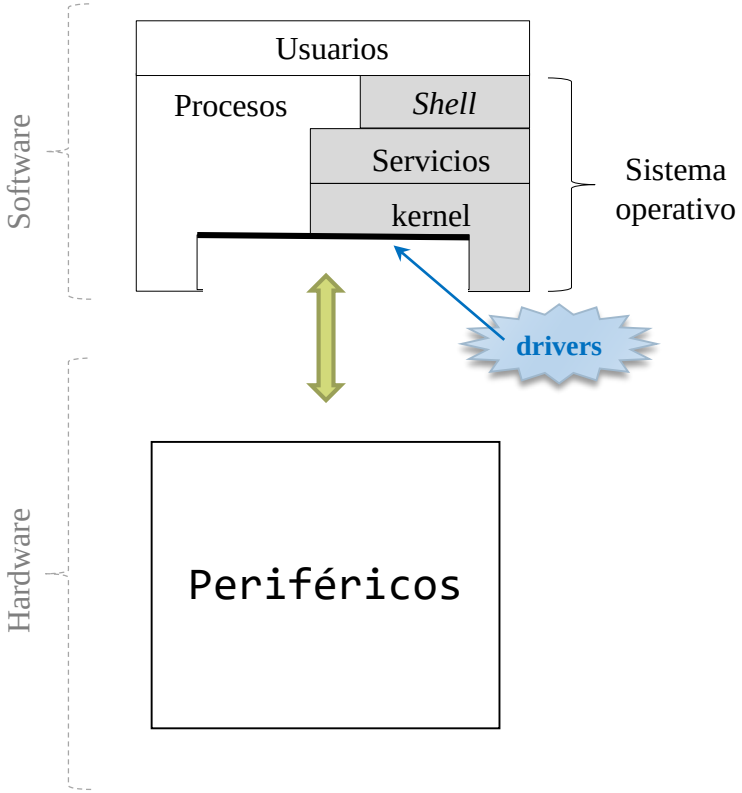
- Introducción
- Organización de los drivers
- Estructura de un driver
- Ejemplos de diseño

Contenidos



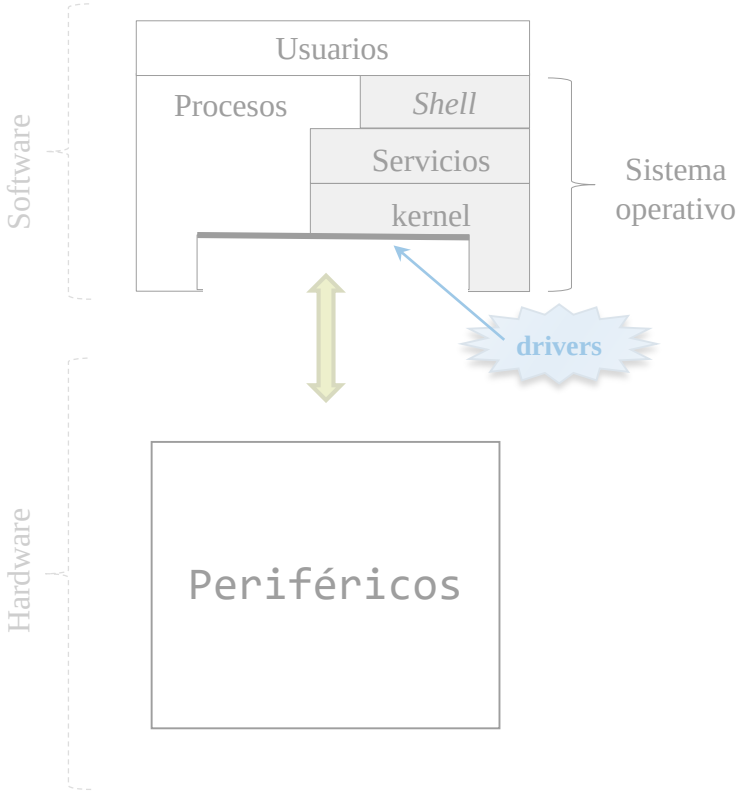
- **Introducción**
- Organización de los drivers
- Estructura de un driver
- Ejemplos de diseño

Ámbito de gestión



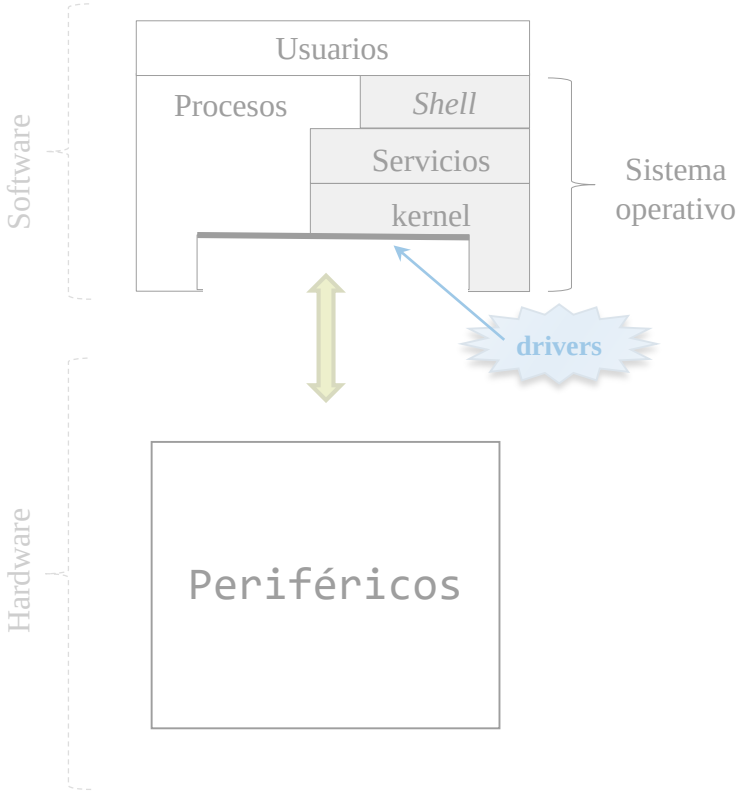
- Parte del sistema operativo encargada de la interacción con todos los posibles controladores (hardware)
- Incluye toda la comunicación de la CPU y la memoria con el resto de elementos hardware.

Características por el ámbito de gestión



- Dependiente del sistema operativo:
  - Los controladores de un sistema operativo no son fáciles de reutilizar en otro.
- Parte muy dinámica:
  - Se añade drivers continuamente.
- Implementados en módulos:
  - Añadir/quitar sin parar.

Objetivos de la E/S



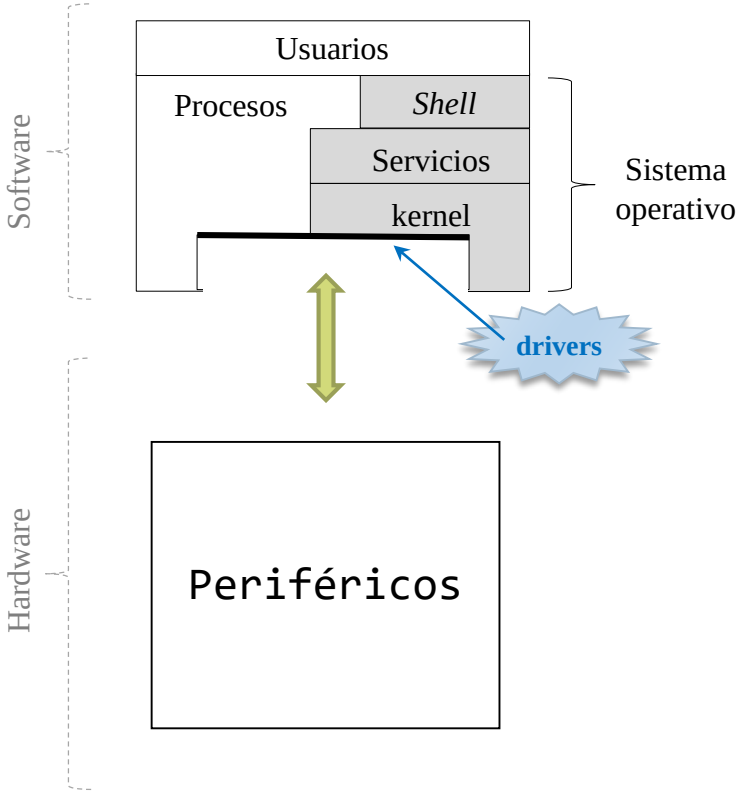
- Ofrecer una visión lógica simplificada para:
  - Resto del sistema operativo
  - Usuarios
- Optimizar la E/S
- Facilitar la gestión de periféricos
- Facilitar añadir soporte a nuevos dispositivos

Importancia de los controladores

- Estadísticas del kernel de Linux (2007-2008):
  - 9,2 millones de líneas de código.
  - Se incrementa un 10% cada año:
  - La mayor parte del código es de los drivers:
    - El 55% del código son los controladores de dispositivo (o drivers)
      - Software parte del sistema operativo que la CPU ejecuta para trabajar con el dispositivo asociado
      - Es un código que funciona con acceso total al sistema (mismo nivel de protección que el kernel)
    - El núcleo del kernel ocupa un 5% y el resto (40%) se reparte entre soporte para las distintas arquitectura, el código de red, etc.

http://cityblogger.com/archives/2008/06/16/linux-kernel-stats

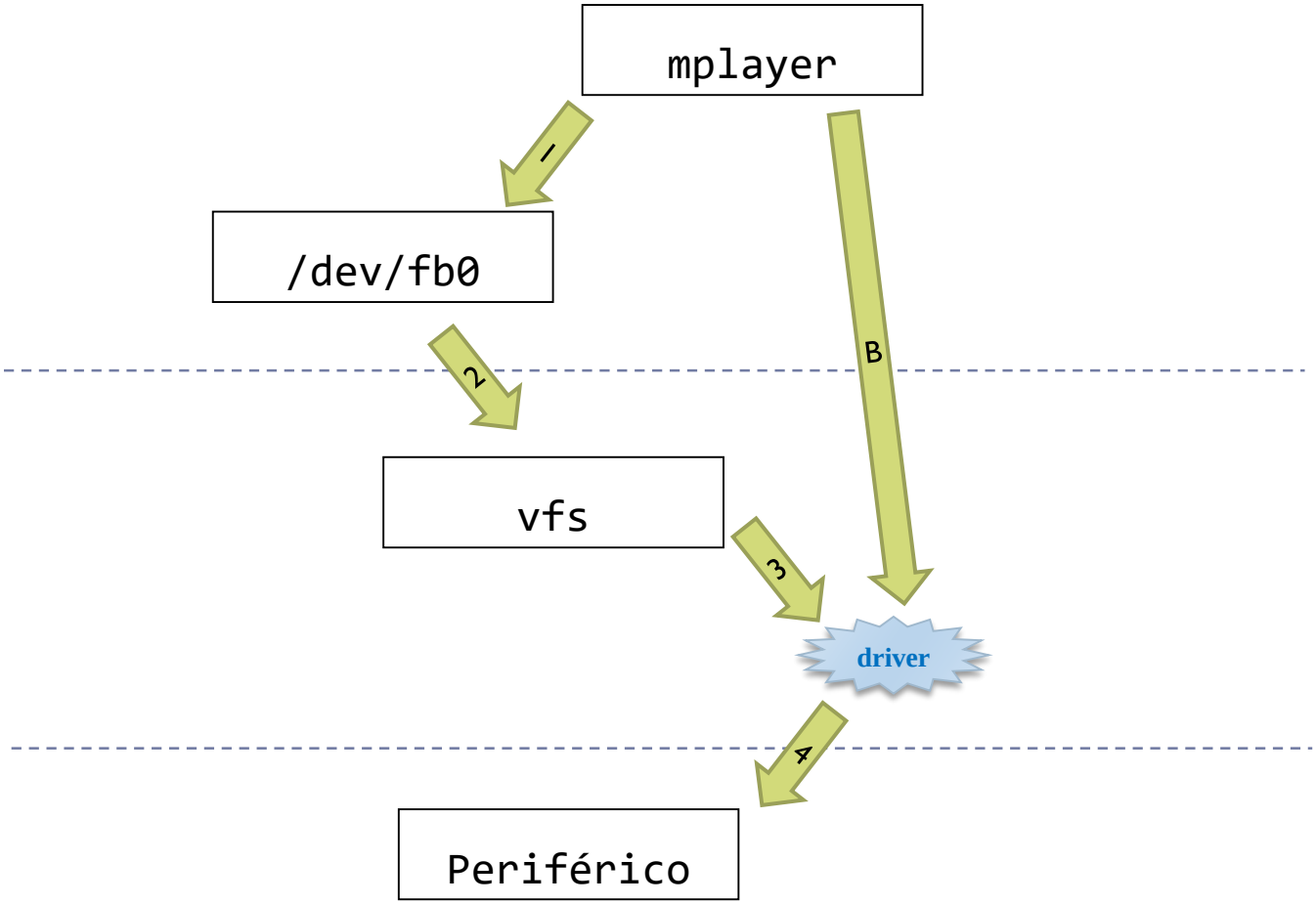
Contenidos



- Introducción
- **Organización de los drivers**
- Estructura de un driver
- Ejemplos de diseño

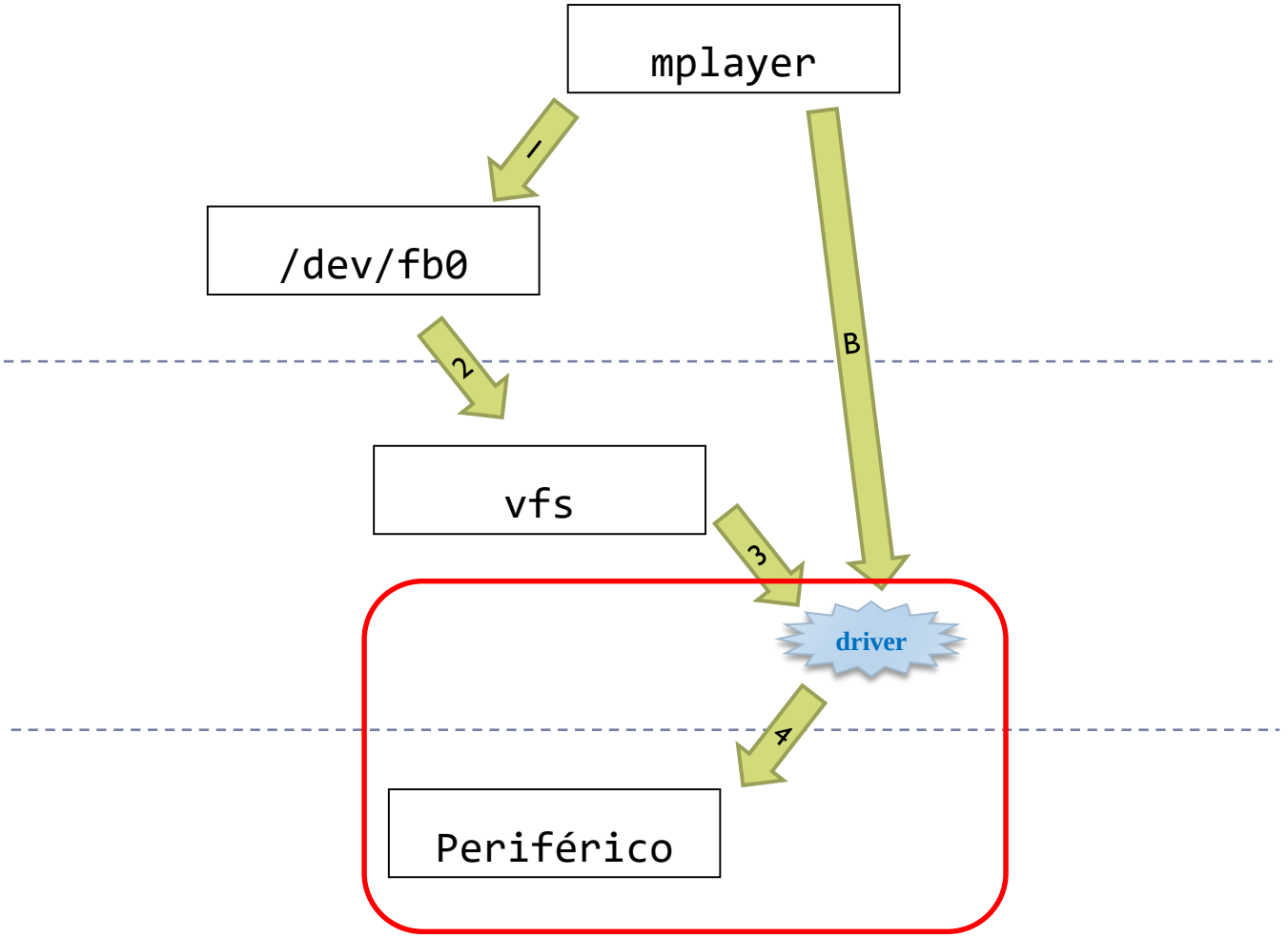
Visión lógica simplificada

Linux



Visión lógica simplificada

Linux



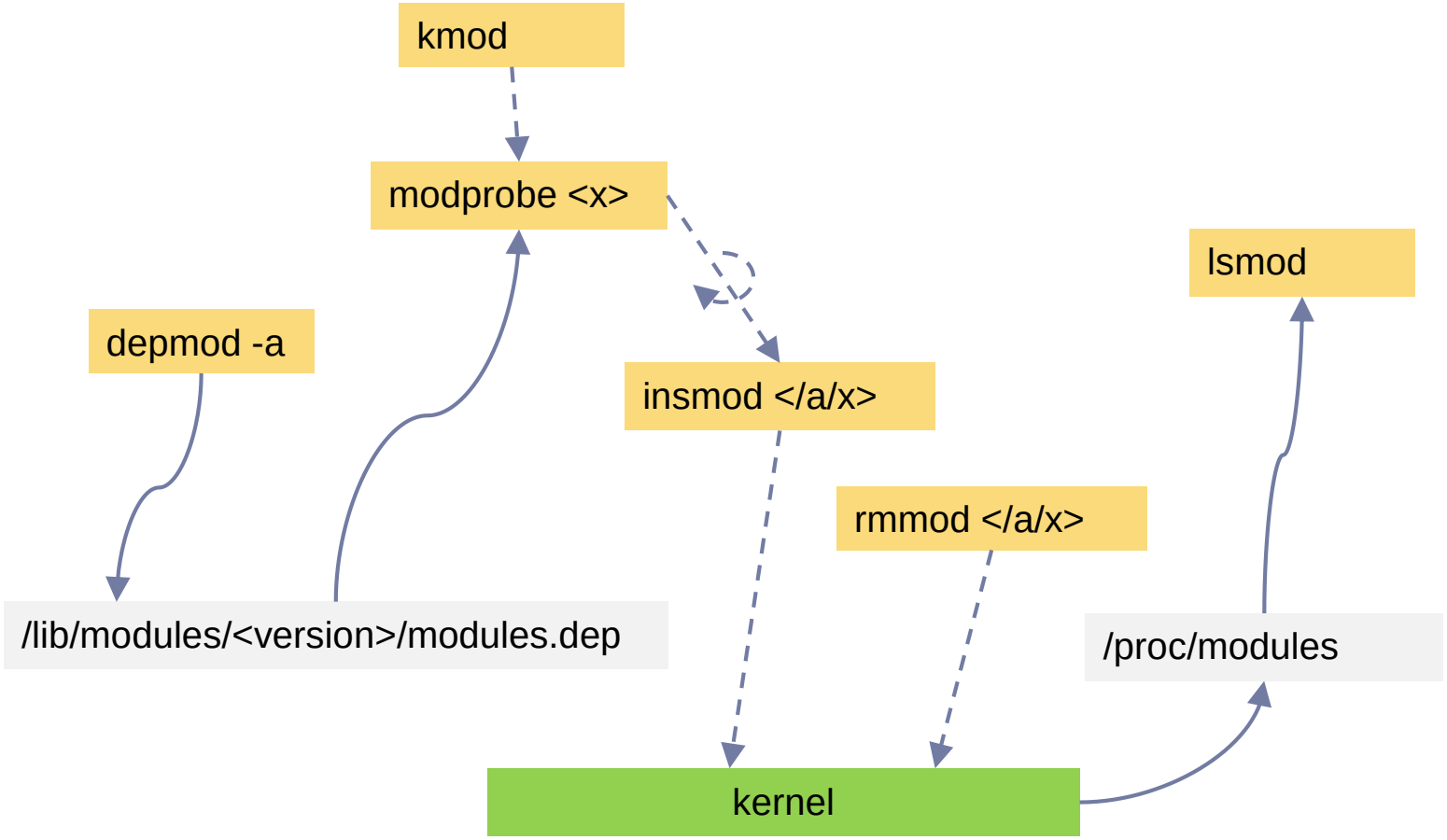
Inventario de hardware
Linux

- Al arrancar el kernel descubre los periféricos.
  - Les asocia el driver más apropiado que disponga.
  - *Hotplugging* (Inserción en caliente de hardware)

```
alejandro@tesla:~$ lspci
00:00.0 Host bridge: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor DRAM Controller (rev 09)
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor PCI Express Root Port (rev 09)
00:02.0 VGA compatible controller: Intel Corporation Xeon E3-1200 v2/3rd Gen Core processor Graphics Controller (rev 09)
00:1a.0 USB controller: Intel Corporation 6 Series/C200 Series Chipset Family USB Enhanced Host Controller #2 (rev 05)
00:1b.0 Audio device: Intel Corporation 6 Series/C200 Series Chipset Family High Definition Audio Controller (rev 05)
…
alejandro@tesla:~$ lsusb
Bus 002 Device 004: ID 046d:c52b Logitech, Inc. Unifying Receiver
Bus 002 Device 005: ID 046d:082b Logitech, Inc.
Bus 002 Device 003: ID 04cc:1521 ST-Ericsson USB 2.0 Hub
Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
…
alejandro@tesla:~$ lshw
…
```

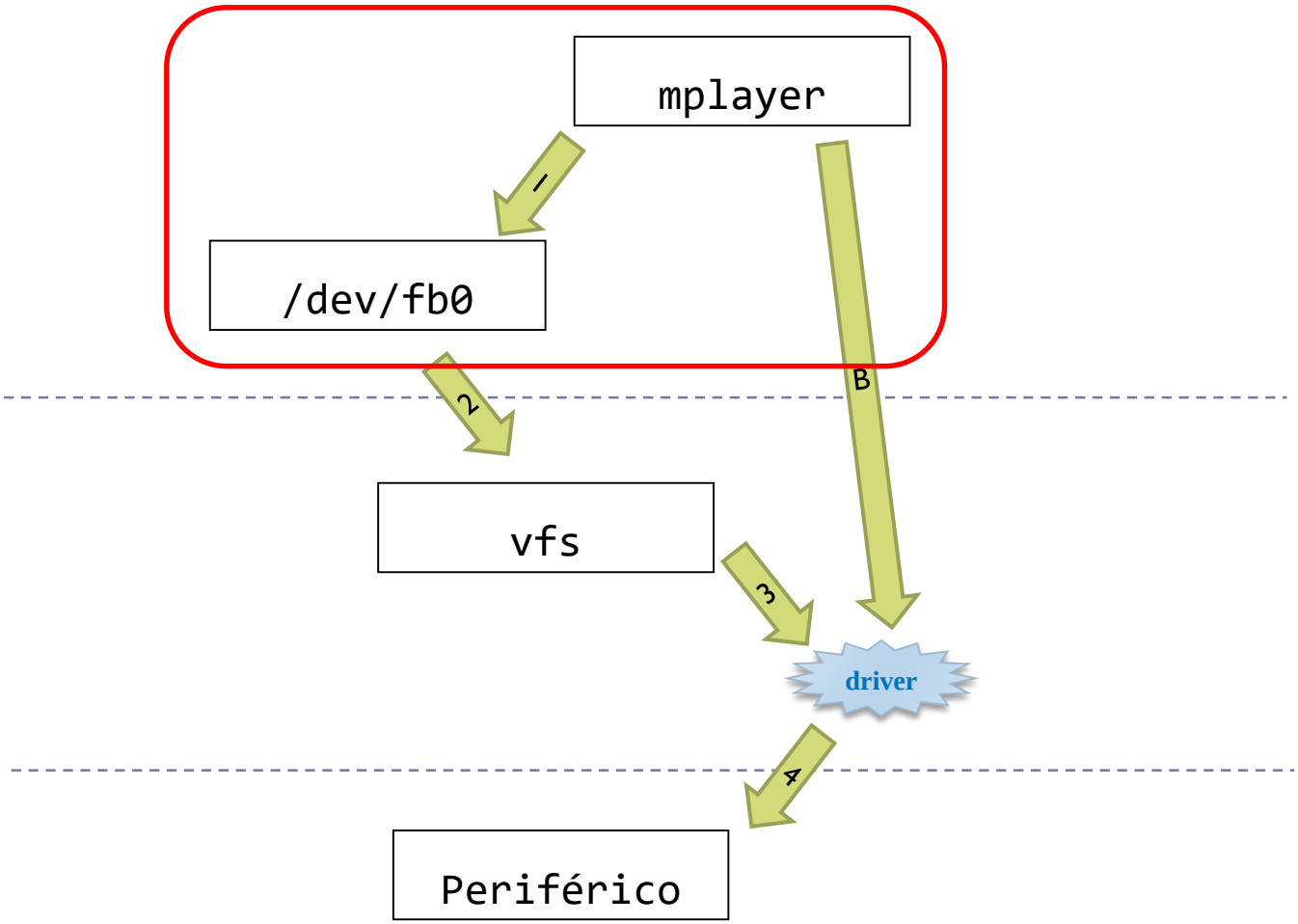
Gestión básica de drivers

Linux



Visión lógica simplificada

Linux



Representación de los dispositivos
Linux

- Habitualmente como ficheros de /dev: `/dev/xxxx`
  - Puede existir ficheros para dispositivos virtuales y dispositivos para los que no hay fichero:
    Tarjeta de red, entrada o salida estándar, etc.

```
alejandro@tesla:~$ ls -las /dev/
total 4
0 crw-------   1 root root       5,   1 feb 16 12:59 console
0 crw-rw----   1 root video     29,   0 feb 16 12:59 fb0
0 crw-r-----   1 root kmem       1,   1 feb 16 12:59 mem
0 crw-rw-rw-   1 root root       1,   3 feb 16 12:59 null
0 crw-------   1 root root      10,   1 feb 16 12:59 psaux
0 brw-rw----   1 root disk       1,   0 feb 16 12:59 ram0
0 crw-rw-rw-   1 root root       1,   8 feb 16 12:59 random
0 crw-------   1 root root     254,   0 feb 16 12:59 rtc0
0 brw-rw----   1 root disk       8,   0 feb 16 12:59 sda
0 brw-rw----   1 root disk       8,   1 feb 16 12:59 sda1
0 brw-rw----   1 root disk       8,   2 feb 16 12:59 sda2
0 crw-rw-rw-   1 root tty        5,   0 feb 20 20:30 tty
0 crw-rw-rw-   1 root root       1,   9 feb 16 12:59 urandom
0 crw-rw-rw-   1 root root       1,   5 feb 16 12:59 zero
…
```


http://sopa.dis.ulpgc.es/ii-dso/leclinux/drivers/i0/io.pdf

Representación de los dispositivos
Linux

- Se identifican con:
  - *Major number* (driver) + *minor number* ("dispositivo")

```
alejandro@tesla:~$ ls -las /dev/
total 4
0 crw-------   1 root root       5,   1 feb 16 12:59 console
0 crw-rw----   1 root video     29,   0 feb 16 12:59 fb0
0 crw-r-----   1 root kmem       1,   1 feb 16 12:59 mem
0 crw-rw-rw-   1 root root       1,   3 feb 16 12:59 null
0 crw-------   1 root root      10,   1 feb 16 12:59 psaux
0 brw-rw----   1 root disk       1,   0 feb 16 12:59 ram0
0 crw-rw-rw-   1 root root       1,   8 feb 16 12:59 random
0 crw-------   1 root root     254,   0 feb 16 12:59 rtc0
0 brw-rw----   1 root disk       8,   0 feb 16 12:59 sda
0 brw-rw----   1 root disk       8,   1 feb 16 12:59 sda1
0 brw-rw----   1 root disk       8,   2 feb 16 12:59 sda2
0 crw-rw-rw-   1 root tty        5,   0 feb 20 20:30 tty
0 crw-rw-rw-   1 root root       1,   9 feb 16 12:59 urandom
0 crw-rw-rw-   1 root root       1,   5 feb 16 12:59 zero
…
```


http://sopa.dis.ulpgc.es/ii-dso/leclinux/drivers/i0/io.pdf

Representación de los dispositivos
Linux

- Gestionado mediante:
  - mkdev (obsoleto): script para creación de todos los posibles ficheros
  - devfs (obsoleto): sistema de ficheros con todos los posibles dispositivos
  - udev: sistema de ficheros dinámico (*hot-plug/unplug*, *triggers*, etc.)

```
alejandro@tesla:~$ ls -las /dev/
total 4
0 crw-------   1 root root       5,   1 feb 16 12:59 console
0 crw-rw----   1 root video     29,   0 feb 16 12:59 fb0
0 crw-r-----   1 root kmem       1,   1 feb 16 12:59 mem
0 crw-rw-rw-   1 root root       1,   3 feb 16 12:59 null
0 crw-------   1 root root      10,   1 feb 16 12:59 psaux
0 brw-rw----   1 root disk       1,   0 feb 16 12:59 ram0
0 crw-rw-rw-   1 root root       1,   8 feb 16 12:59 random
0 crw-------   1 root root     254,   0 feb 16 12:59 rtc0
0 brw-rw----   1 root disk       8,   0 feb 16 12:59 sda
0 brw-rw----   1 root disk       8,   1 feb 16 12:59 sda1
0 brw-rw----   1 root disk       8,   2 feb 16 12:59 sda2
0 crw-rw-rw-   1 root tty        5,   0 feb 20 20:30 tty
0 crw-rw-rw-   1 root root       1,   9 feb 16 12:59 urandom
0 crw-rw-rw-   1 root root       1,   5 feb 16 12:59 zero
...
```


http://sopa.dis.ulpgc.es/ii-dso/leclinux/drivers/i0/io.pdf

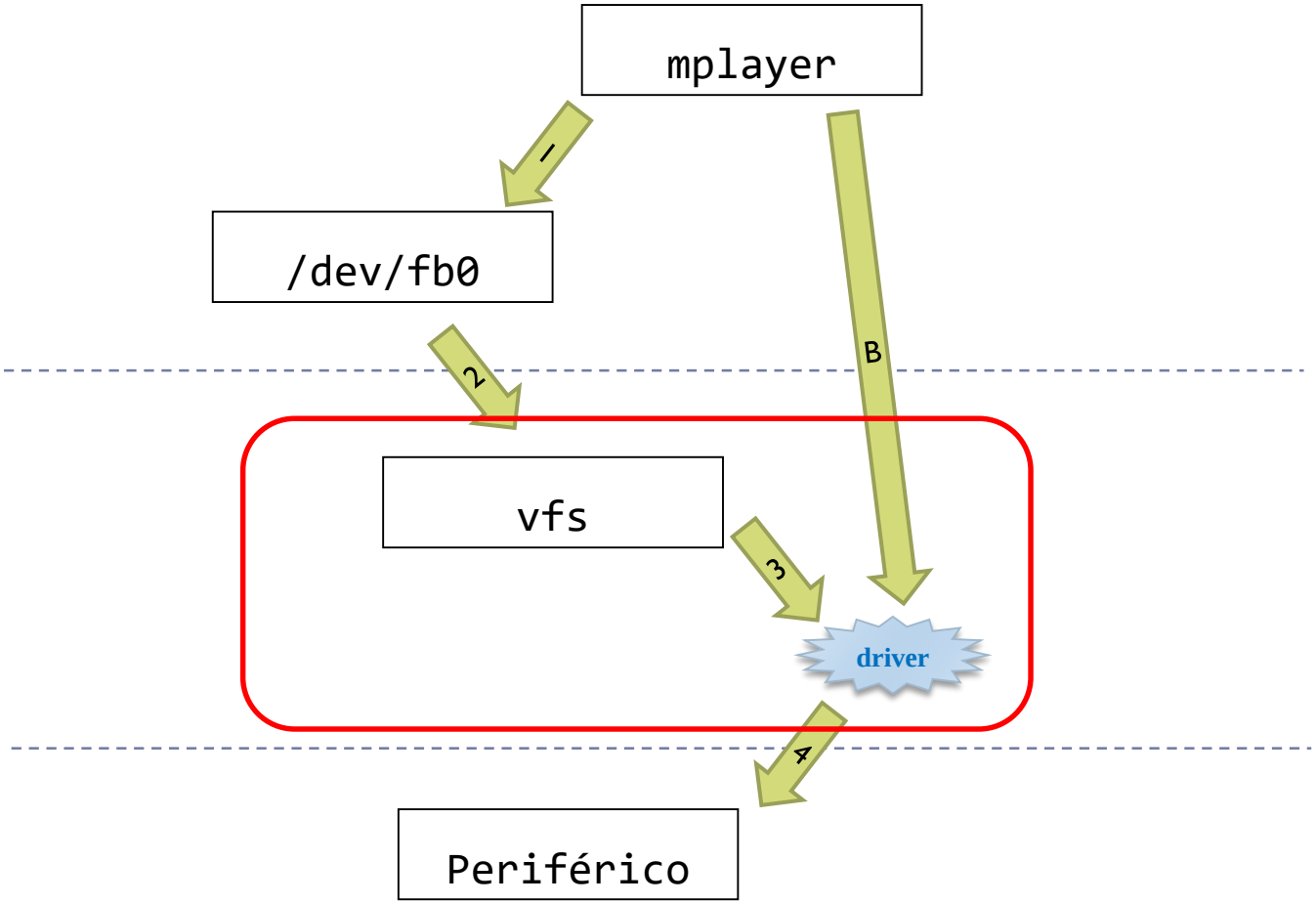
Representación de los dispositivos
Linux

- Es posible manualmente crear un nuevo fichero de dispositivo:
  - Nombre del fichero
  - Tipo: bloque o carácter
  - *Major & minor number*

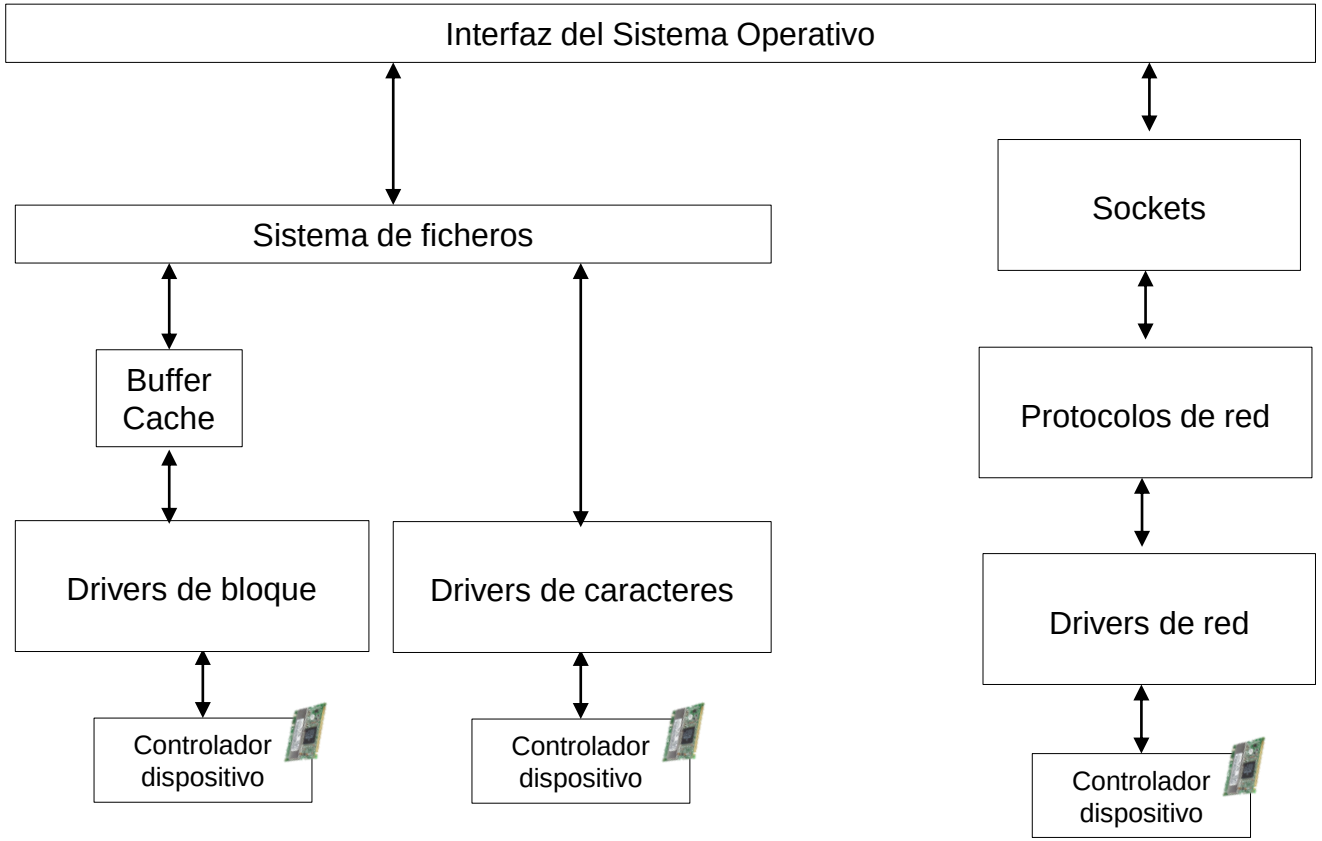
```
alejandro@tesla:~$ mknod /dev/sensor1 c 12 1

alejandro@tesla:~$ ls sensor1
0 crw-r--r--   1 root root      12,   1 feb 21 13:46 sensor1

alejandro@tesla:~$ cat sensor1
cat: /dev/sensor1: No existe el dispositivo o la dirección

alejandro@tesla:~$ udevadm info -a -n /dev/sda | grep DRIVER
    DRIVERS=="sd"
    DRIVERS=="ata_piix“
```

Visión lógica simplificada

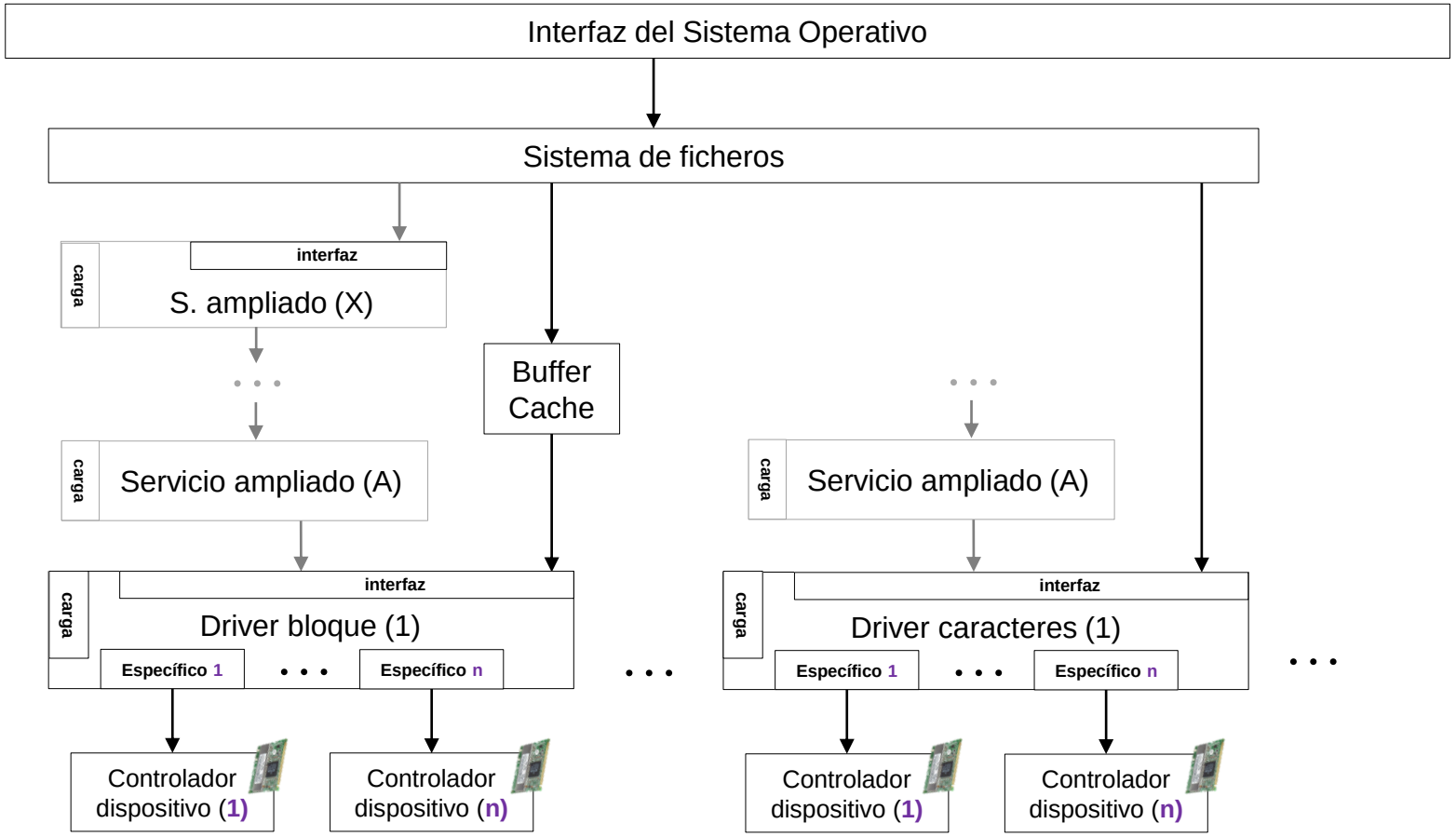
Linux



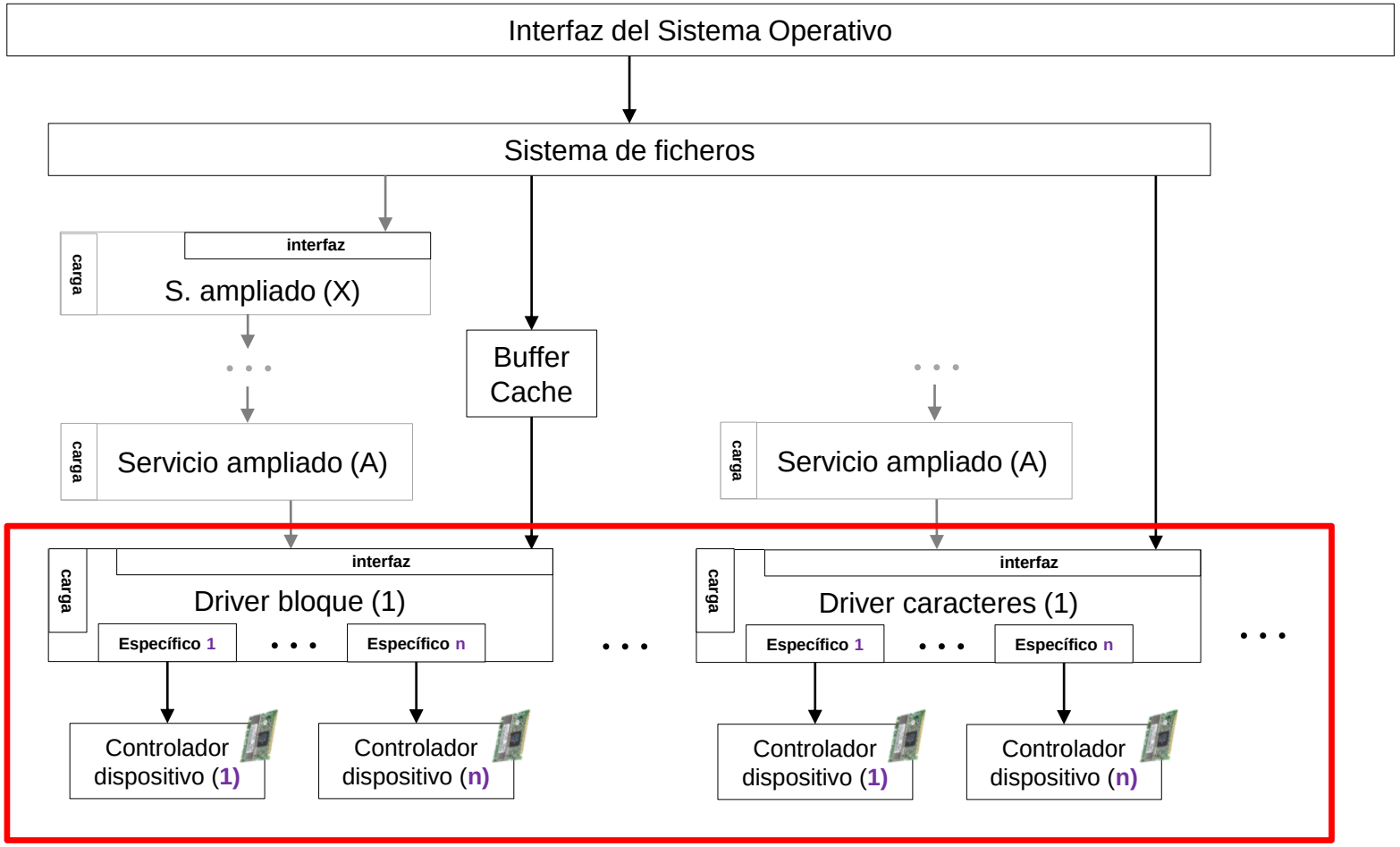
Arquitectura del sistema de E/S



Estructura genérica del sistema de E/S

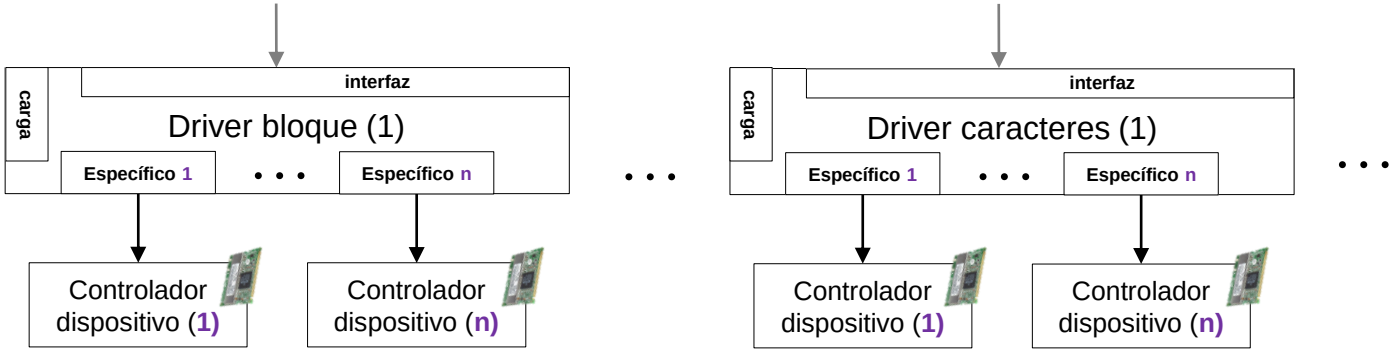


Estructura genérica del sistema de E/S

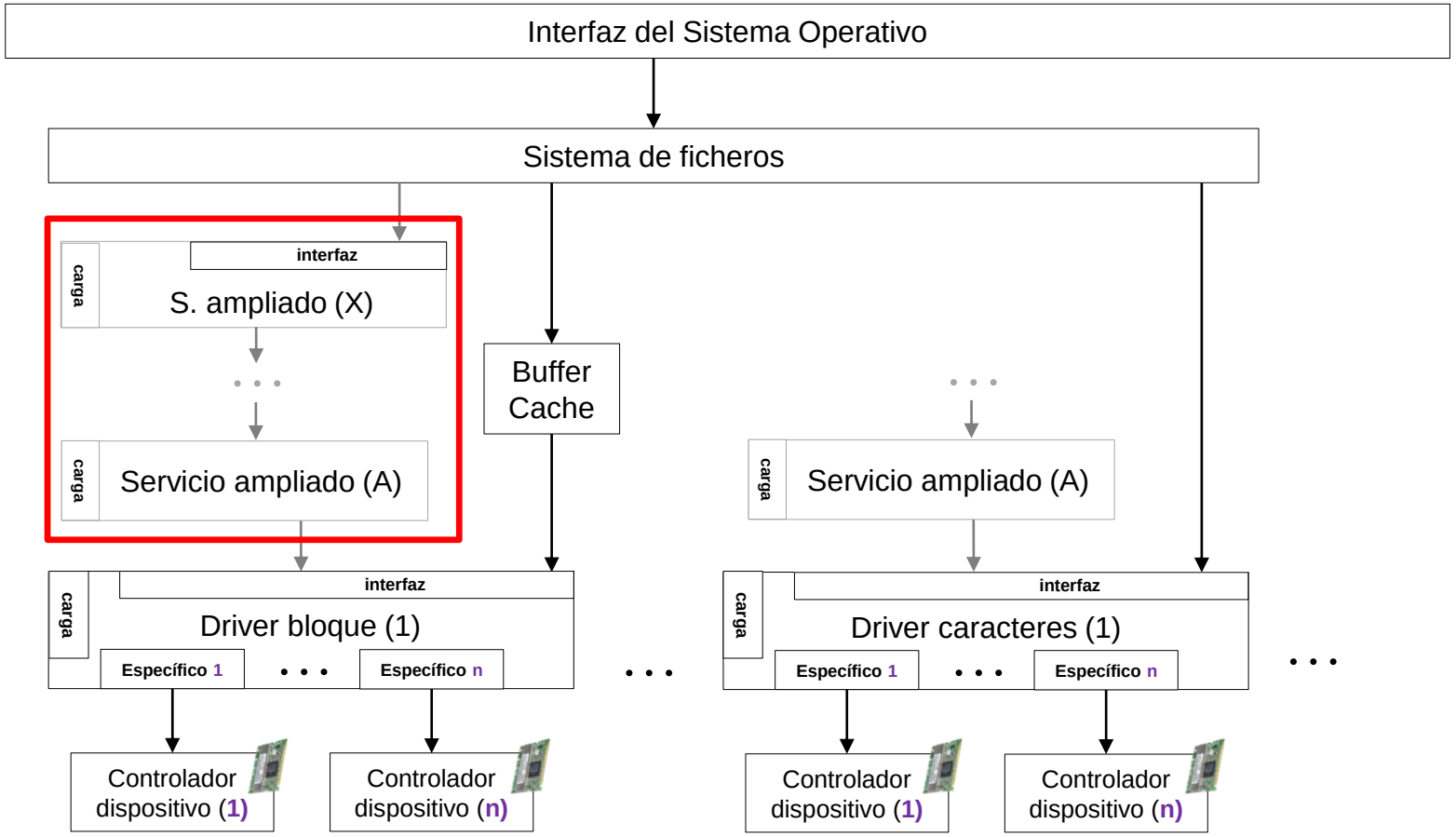


Estructura genérica del sistema de E/S
clasificación de drivers

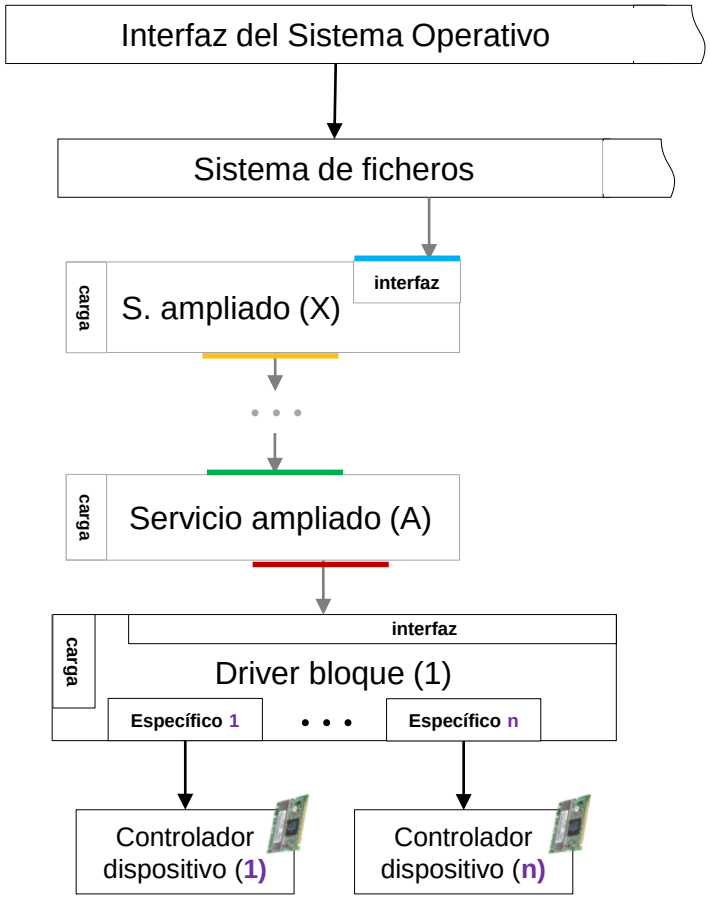
- Los drivers se clasifican según el grupo de dispositivos a los que trata.
  - Si dos drivers tratan un mismo tipo de dispositivo entonces la interfaz es similar
  - Parte de la implementación del driver es común (se ahorra código)
- De forma clásica hay tres tipos:
  - Dispositivos de caracteres: teclado, módem, etc.
  - Dispositivos de bloques: discos, cintas, etc.
  - Dispositivos de red: tarjetas de red



Estructura genérica del sistema de E/S

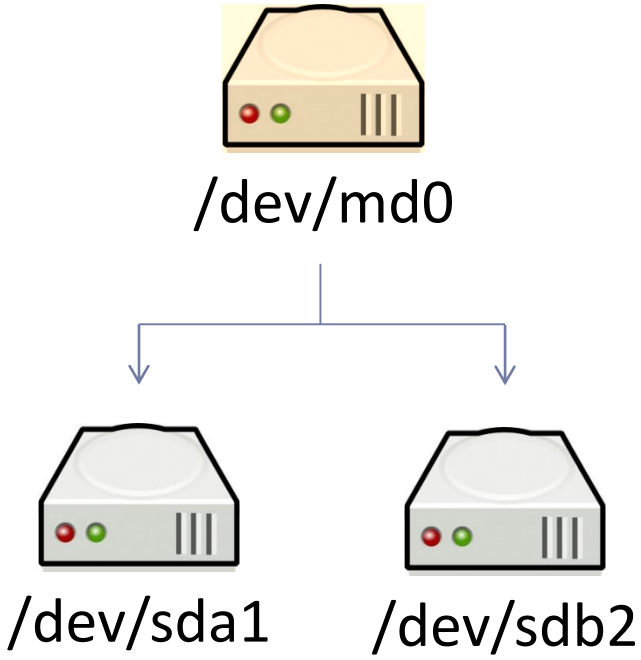


Estructura genérica del sistema de E/S
servicios ampliados



- Servicio ampliado:
  - Módulo que extiende un driver para añadirle algún tipo de funcionalidad.
  - Son apilables entre sí.
- Tiene, al menos, dos interfaces:
  - La interfaz del servicio que ofrece:
    - Interfaz de llamadas al sistema
    - Interfaz de un S. Ampliado superior
  - La interfaz del recurso que utiliza:
    - Interfaz de un driver
    - Interfaz de un S. Ampliado inferior

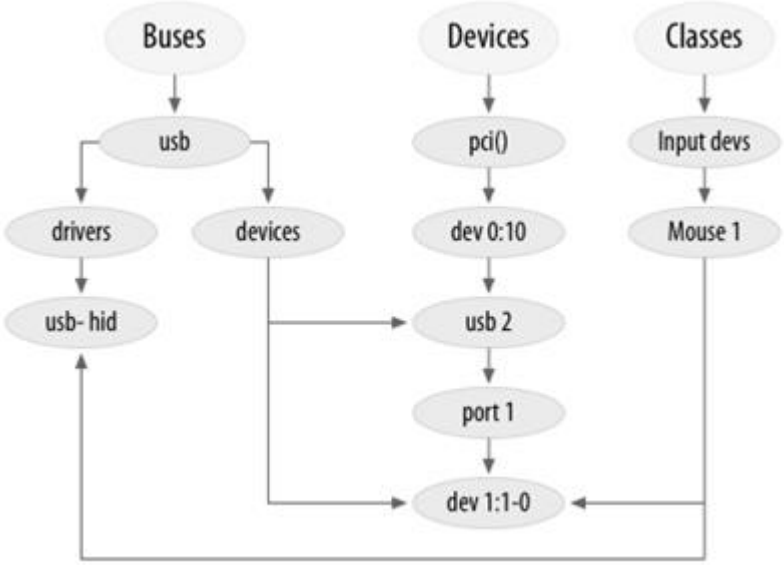
Servicios ampliados
Linux



- Ejemplo de servicio ampliado:
  - md (*multiple disks*)
- Combina varios discos duros, o particiones (o volúmenes) en un único disco virtual.
  - `mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sda1 /dev/sdb1`

Jerarquía de drivers
Linux

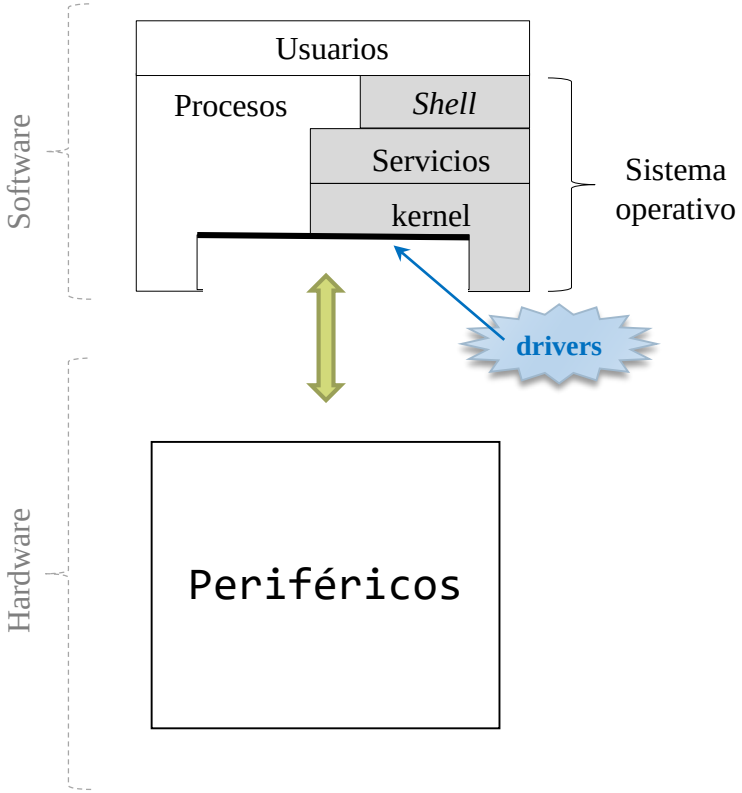
- La jerarquía del modelo se muestra en la figura:
  - Buses en el nivel inferior
  - Dispositivos en el nivel intermedio
  - Clases en el nivel más alto



- Acceso a través de sysfs:
  - /sys/block: disp. de bloques (cualquier bus)
  - /sys/bus: buses del sistema (disp. están aquí)
  - /sys/devices: dispositivos organizados por buses
  - /sys/class: clases de dispositivos (audio, de red, ...)
  - /sys/module: drivers registrados en el núcleo
  - /sys/power: manejo del estado de energía
  - /sys/firmware: manejo de firmware (en ciertos disp.)

http://sopa.dis.ulpgc.es/ii-dso/leclinux/drivers/i0/io.pdf

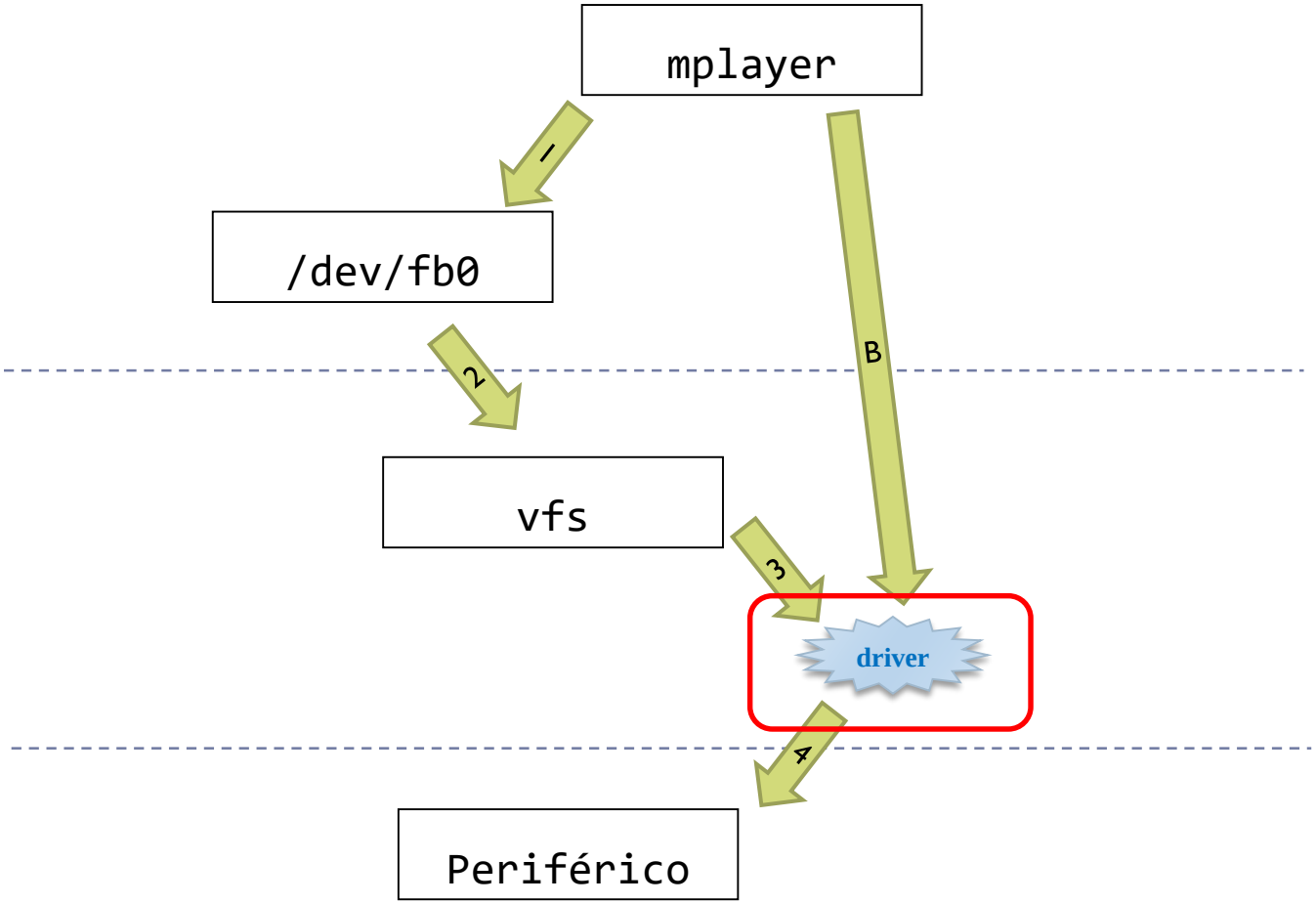
Contenidos



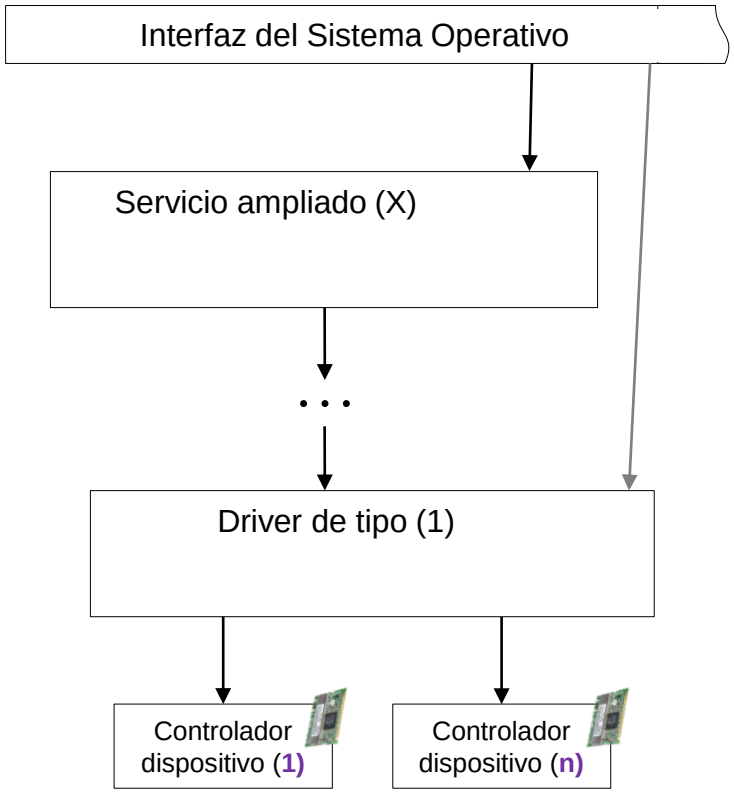
- Introducción
- Organización de los drivers
- **Estructura de un driver**
- Ejemplos de diseño

Visión lógica simplificada

Linux



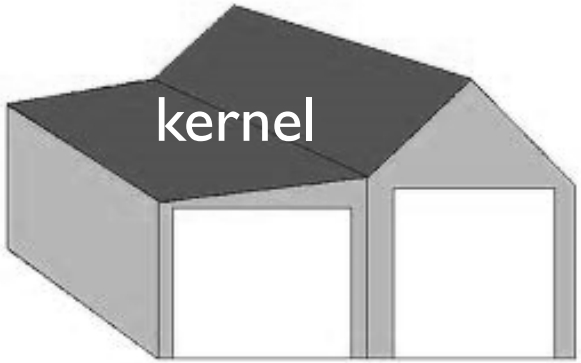
Organización básica de un driver/s.a.
drivers basados en módulos del kernel

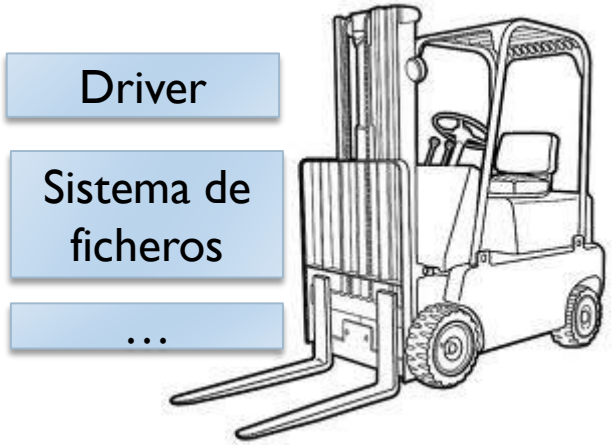


- No todos los drivers/ss.aa. son necesarios en todo momento:
  - Hay dispositivos que se conectan/desconectan sin apagar el ordenador (*hot-plug*)
- Hay dos métodos (combinables) para la selección de drivers/ss.aa. a usar:
  - Elegirlos al compilar el kernel.
    - En el arranque del sistema operativo se crean los drivers/ss.aa. elegidos.
  - Elegirlos mientras el kernel está ejecutando (enlace dinámico).
    - Se crean en algún punto de la ejecución del sistema operativo.

Módulos para ampliar el kernel

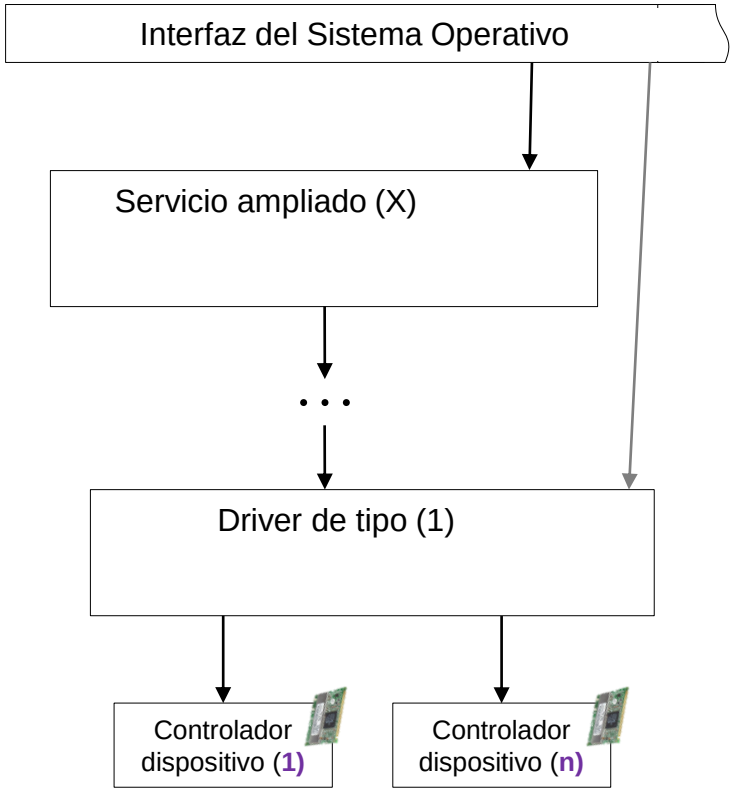
- Los módulos se utilizan no solo para los drivers de los dispositivos, actualmente también se utilizan para añadir otro tipos de funcionalidad:
  - Sistemas de ficheros, protocolos de red, llamadas al sistema extras, etc.





Organización básica de un driver/s.a.

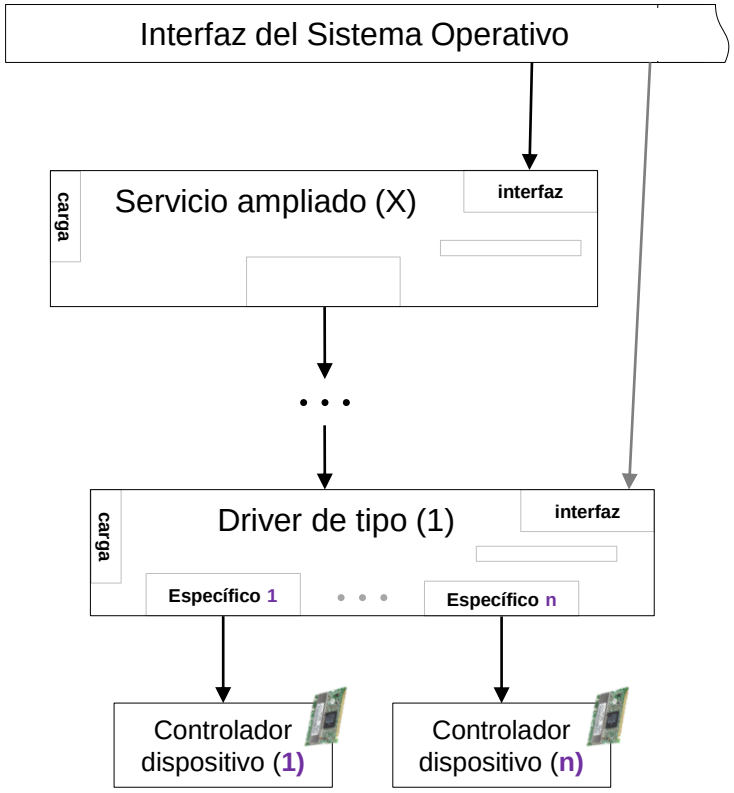
partes internas de un módulo driver




```
/* modulo_teclado.c (Javier Fernández Muñoz) */

#include <string.h>
#include <stdlib.h>
#include "minikernel.h"


/*  Tipo con atributos específicos
    del dispositivo de teclado          */
typedef struct  {
     TipoBufferCaracteres bufferCaracteres;
     TipoListaBCP listaProcesosBloqueados;
} TipoDatosPropiosDispositivo_teclado;


/*  Descriptor de fichero de teclado  */
int cerrarFichero_teclado (int descFichero) ;
int abrirFichero_teclado   (int descFichero,  char *nombre, int flags) ;
int leerFichero_teclado    (int descFichero, char *buffer, int tamanyo) ;


/*  Dispositivo de teclado  */
int    interrupcionHW_teclado   (int descDispositivo) ;
void  interrupcionSW_teclado    (int descDispositivo) ;
int peticionCaracter_teclado    (int descDispositivo,
                                 char *caracter, int operacion) ;


/* Cargar y descarga de módulos */
int cargarModulo_teclado () ;
int crearDispositivo_teclado  (int descDriver,
                               char *nombreDispositivo, int hardwareID) ;
int destruirDriver_teclado     (int descDriver) ;
int crearDescFicheroDispositivo_teclado (int descDispositivo,
                                 TipoTablaDescFicheros tablaDescFicheros) ;
int mostrarDispositivo_teclado (int descDispositivo,
                                 char *buffer, int bytesLibres) ;
```

Organización básica de un driver/s.a.

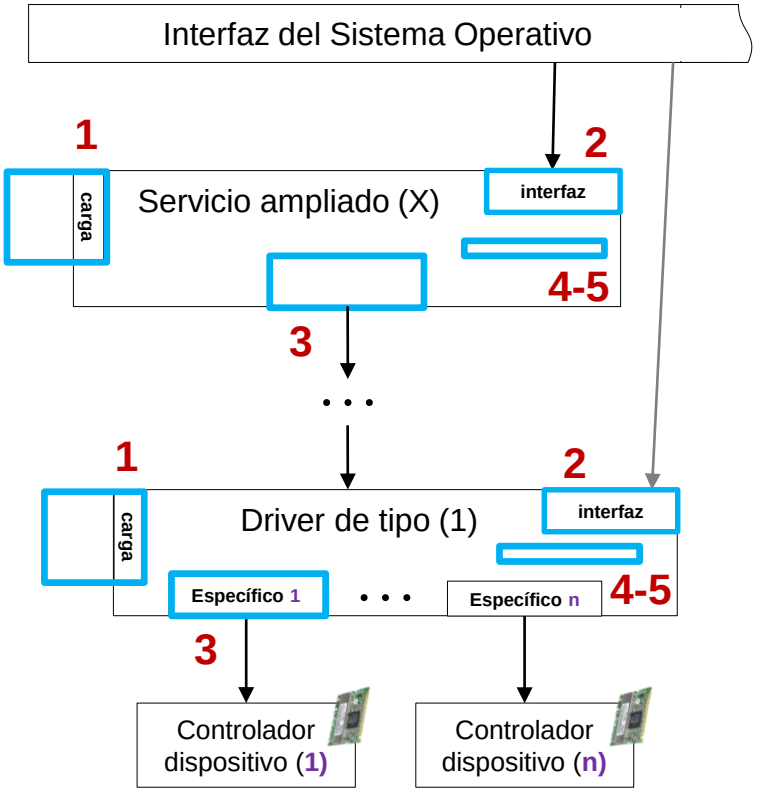
partes internas de un módulo driver




```
/* modulo_teclado.c (Javier Fernández Muñoz) */

#include <string.h>
#include <stdlib.h>
#include "minikernel.h"


/*  Tipo con atributos específicos
    del dispositivo de teclado          */
typedef struct  {
     TipoBufferCaracteres bufferCaracteres;
     TipoListaBCP listaProcesosBloqueados;
} TipoDatosPropiosDispositivo_teclado;


/*  Descriptor de fichero de teclado  */                    interfaz
int cerrarFichero_teclado (int descFichero) ;
int abrirFichero_teclado   (int descFichero,  char *nombre, int flags) ;
int leerFichero_teclado    (int descFichero, char *buffer, int tamanyo) ;

/*  Dispositivo de teclado  */                             específico X
int     interrupcionHW_teclado    (int descDispositivo) ;
void  interrupcionSW_teclado    (int descDispositivo) ;
int peticionCaracter_teclado     (int descDispositivo,
                                  char *caracter, int operacion) ;

/* Cargar y descarga de módulos */                          carga
int cargarModulo_teclado () ;
int crearDispositivo_teclado  (int descDriver,
                               char *nombreDispositivo, int hardwareID) ;
int destruirDriver_teclado      (int descDriver) ;
int crearDescFicheroDispositivo_teclado (int descDispositivo,
                                  TipoTablaDescFicheros tablaDescFicheros) ;
int mostrarDispositivo_teclado (int descDispositivo,
                                  char *buffer, int bytesLibres) ;
```

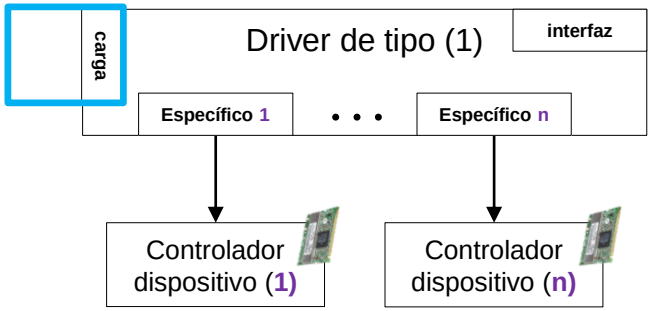
Organización básica de un driver/s.a.

partes internas de un módulo driver



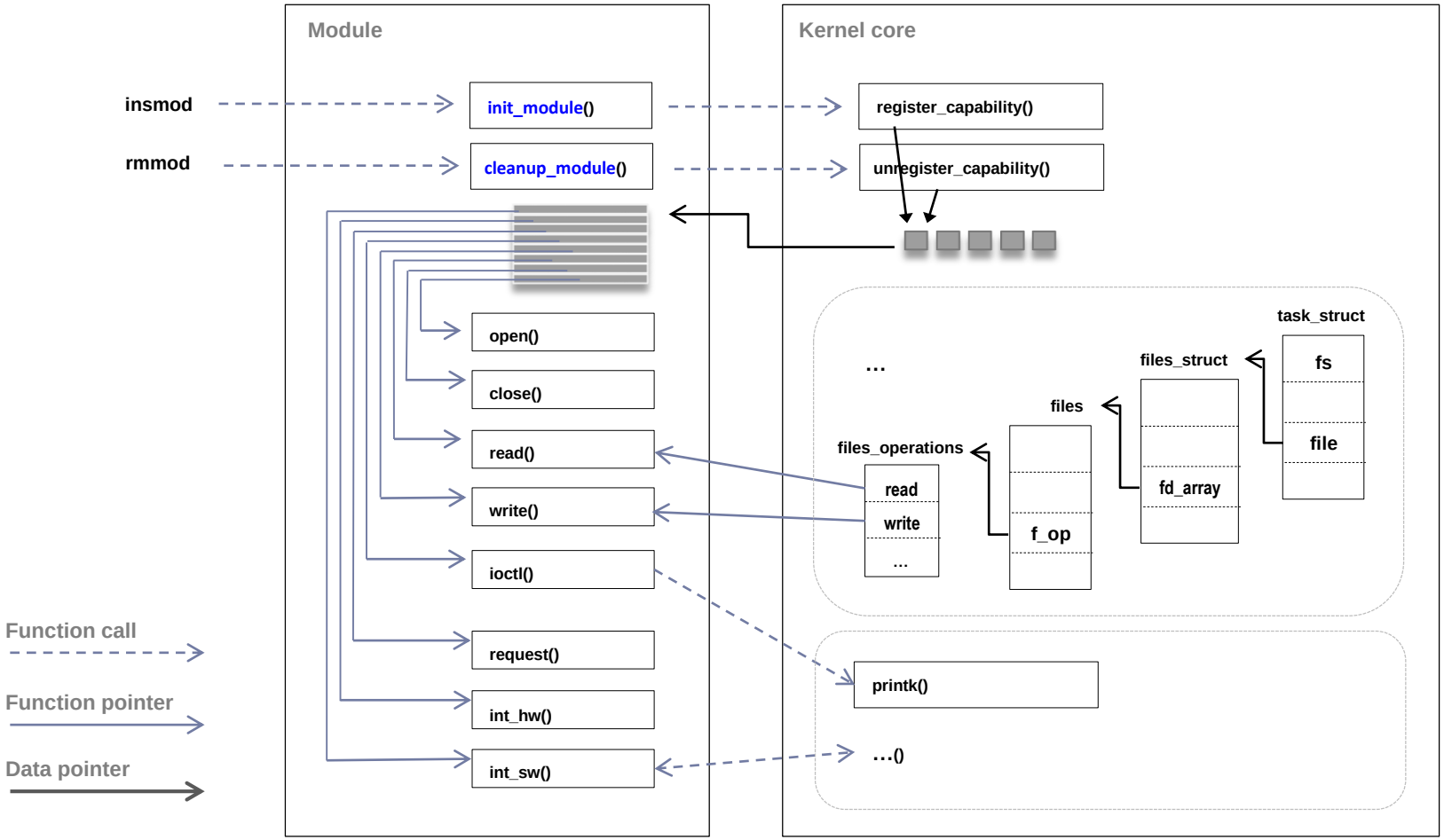
1. Registro de driver
2. Interfaz para llamadas al sistema
3. Petición al controlador de dispositivo
4. Planificación de E/S en el driver
5. Inicialización y finalización del driver

Organización básica
1. registro de drivers



- Tabla con los drivers cargados:
  - Funciones para registrar drivers.
    - cargar el módulo asociado al driver.
  - Funciones para borrar drivers.
- Tabla con dispositivos detectados:
  - Funciones para registrar el dispositivo en el driver, y dar de alta sus estructuras/funciones particulares.
    - Desde el driver se tiene acceso a la lista de dispositivos que maneja.
  - Funciones para buscar y dar de baja un dispositivo.

Estructuras principales de gestión
Linux



http://www.xml.com/ldd/chapter/book/ch02.html

Proceso de registro

Linux

dso/test1.c

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

MODULE_LICENSE("Dual BSD/GPL");

static int hello_init (void)
{
    printk("<1> Test1 cargado…\n");
    return 0;
}

static void hello_exit (void)
{
    printk("<1> Test1 descargado.\n");
}

module_init  (hello_init);
module_exit (hello_exit);
```

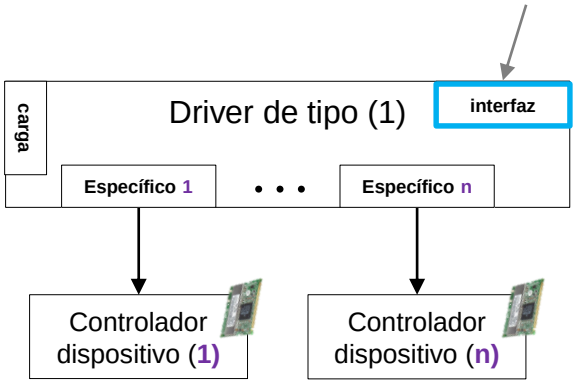
Proceso de registro
Linux

dso/Makefile

```
obj-m := test1.o
```

```
make -C /usr/src/linux M=`pwd` modules
insmod test1.ko
lsmod
dmesg
rmmod test1
```

Organización básica
2. interfaz para llamadas al sistema

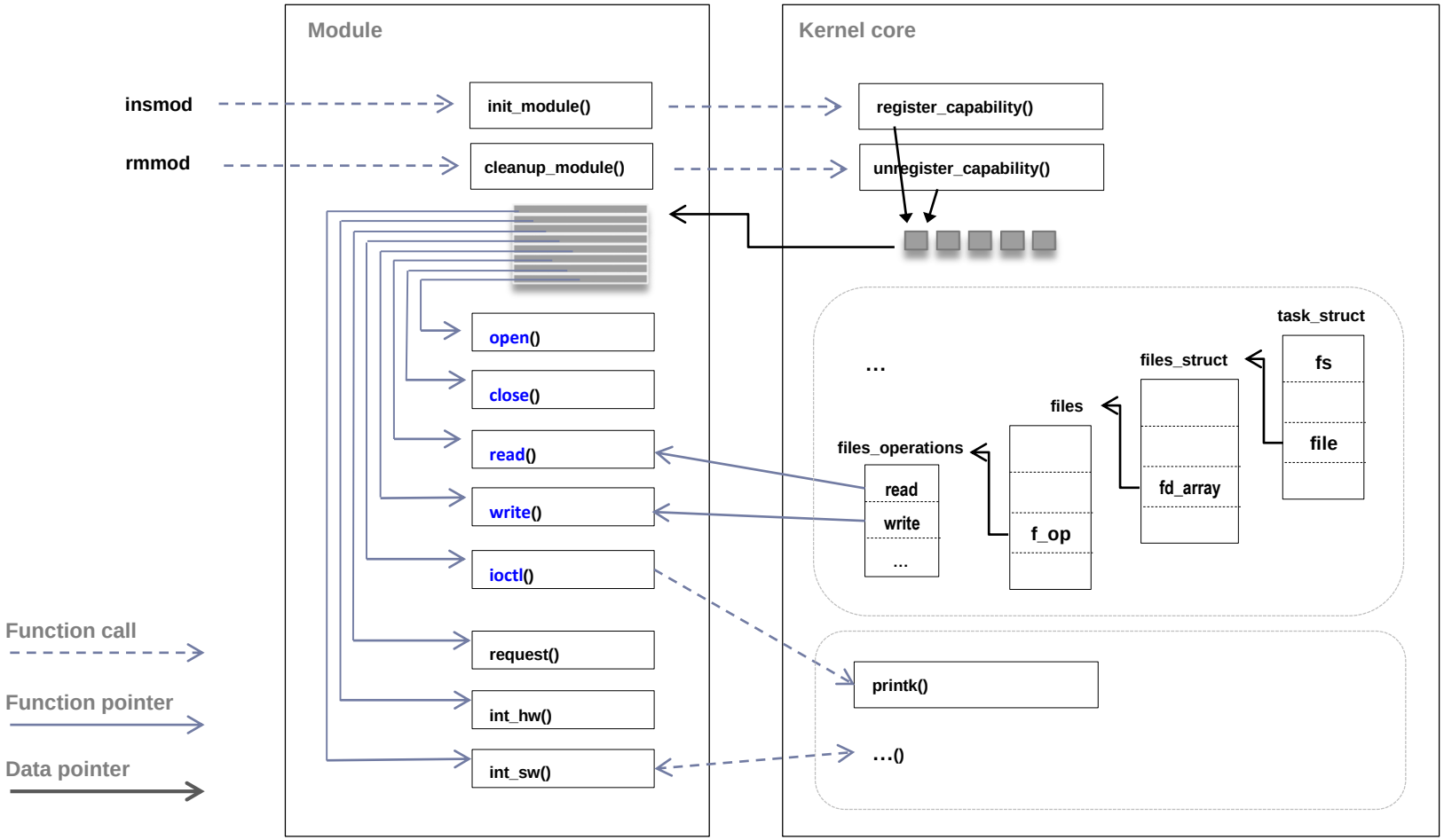


- Interfaz para llamadas al sistema:
  - Conjunto de funciones que proporciona un driver para acceder al dispositivo.
- Características:
  - Estandarización:
    - Si un dispositivos hardware es válido para una tarea, el programa de usuario o servicio del SO que la realiza debe poder utilizarlo sin modificar su código.
  - Uso de interfaces comunes y reducidas de llamadas al sistema:
    - Crear una nueva llamada es más costoso que reutilizar llamadas ya existentes.

Interfaz para llamadas al sistema
Linux

- Llamadas para establecer el acceso al dispositivo:
  - Open (nombre, flags, modo)
  - Close (descriptor)
- Llamadas para intercambiar datos con el dispositivo:
  - Read (descriptor, buffer, tamaño)
  - Write (descriptor, buffer, tamaño)
  - Lseek (descriptor, desplazamiento, origen)
- Llamadas especificas del dispositivo:
  - Ioctl (descriptor, num_operacion, puntero_parametros)
    - Permite la ejecución de cualquier servicio con cualquier parámetros.
    - Las operaciones debe hacerse públicas de alguna forma para que no haya conflictos entre distintos drivers.

Estructuras principales de gestión
Linux



http://www.xml.com/ldd/chapter/book/ch02.html

Interfaz para llamadas al sistema
Linux

dso/test2.c

```
#include <linux/init.h>
#include <linux/module.h>

#include <linux/kernel.h>      /* printk() */
#include <linux/fs.h>          /* everything... */
#include <linux/errno.h>       /* error codes */
#include <linux/types.h>       /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h>       /* O_ACCMODE */

#include <asm/uaccess.h>  /* copy_from/to_user */


MODULE_LICENSE("Dual BSD/GPL");
```

Interfaz para llamadas al sistema
Linux

dso/test2.c

```
int     test2_open    (struct inode *inode, struct file *filp);
int     test2_release (struct inode *inode, struct file *filp);
ssize_t test2_read    (struct file *filp, char *buf, size_t count, loff_t *f_pos);
ssize_t test2_write   (struct file *filp, const char *buf, size_t count, loff_t *f_pos);

    struct file_operations test2_fops = {
        open:    test2_open,
        release: test2_release, /* AKA close */
        read:    test2_read,
        write:   test2_write
    };

void test2_exit (void);
int  test2_init (void);

    module_init  (test2_init);
    module_exit (test2_exit);
```

Interfaz para llamadas al sistema
Linux

dso/test2.c

```
int test2_major = 60;

int test2_init (void) {
    int result;

    result = register_chrdev (test2_major, ”test2", &test2_fops);
    if (result < 0) {
        printk("<1> test2: error on register_chrdev\n”);
        return result;
    }

    printk("<1>test2: inserted…\n");
    return 0;
}

void test2_exit (void) {
    unregister_chrdev (test2_major, ”test2");
    printk("<1> test2: removed. \n");
}
```

Interfaz para llamadas al sistema
Linux

dso/test2.c

```
int test2_open (struct inode *inode, struct file *filp)
{
    /*
     * Once the associate file is open, increment the usage count
     * Three column from the lsmod output
     */
    try_module_get (THIS_MODULE) ;
    return SUCCESS;
}

int test2_release (struct inode *inode, struct file *filp)
{
    /*
     * Decrement the usage count.
     */
    module_put (THIS_MODULE) ;
    return 0;
}
```

Interfaz para llamadas al sistema
Linux

dso/test2.c

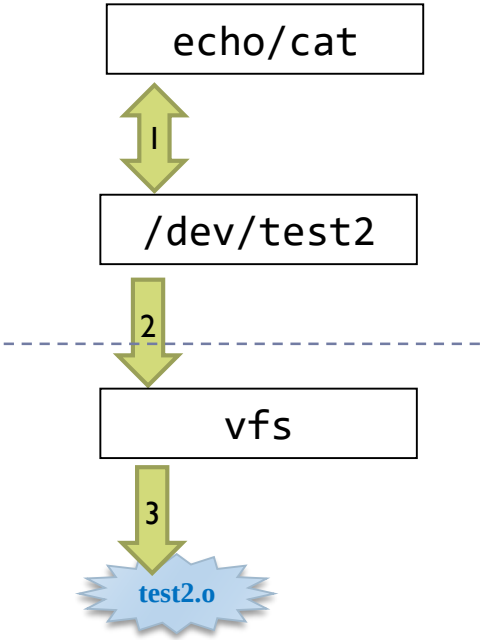
```
char  test2_buffer = ‘a’;

ssize_t test2_read (struct file *filp, char *buf, size_t count, loff_t *f_pos)
{
    if (*f_pos > 1024) {
        return 0;
    }

    copy_to_user (buf, &test2_buffer, 1);
    *f_pos+=1;
    return 1;
}

ssize_t test2_write ( struct file *filp, const char *buf, size_t count, loff_t *f_pos )
{
     copy_from_user (&test2_buffer, buf,1);
     return 1;
}
```

Interfaz para llamadas al sistema
Linux

dso/Makefile

```
obj-m := test1.o test2.o
```

```
make -C /usr/src/linux M=`pwd` modules
insmod test2.ko
dmesg

mknod /dev/test2 c 60 0
chmod 777    /dev/test2
echo –n ‘b’ >  /dev/test2
cat /dev/test2

rm –fr /dev/test2
rmmod test2
```



Interfaz para llamadas al sistema
Linux

- Llamadas para establecer el acceso al dispositivo:
  - Open (nombre, flags, modo)
  - Close (descriptor)
- Llamadas para intercambiar datos con el dispositivo:
  - Read (descriptor, buffer, tamaño)
  - Write (descriptor, buffer, tamaño)
  - Lseek (descriptor, desplazamiento, origen)

+

Iread
Iwrite
Wait
Ready

- Llamadas especificas del dispositivo:
  - Ioctl (descriptor, num_operacion, puntero_parametros)
    - Permite la ejecución de cualquier servicio con cualquier parámetros.
    - Las operaciones debe hacerse públicas de alguna forma para que no haya conflictos entre distintos drivers.

http://www.ibm.com/developerworks/linux/library/l-async/

Interfaz para llamadas al sistema

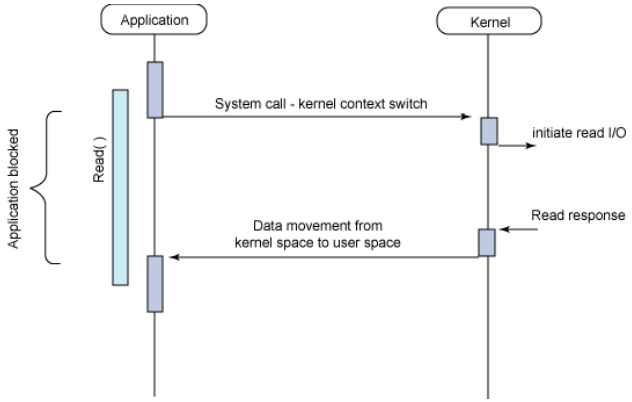
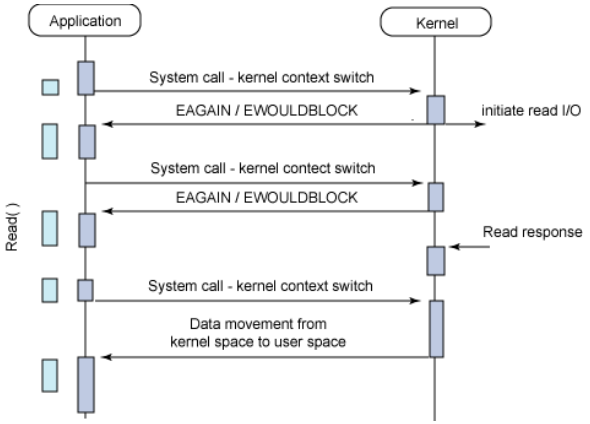
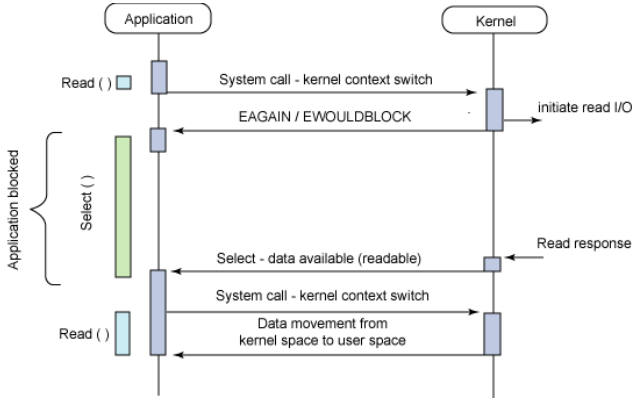
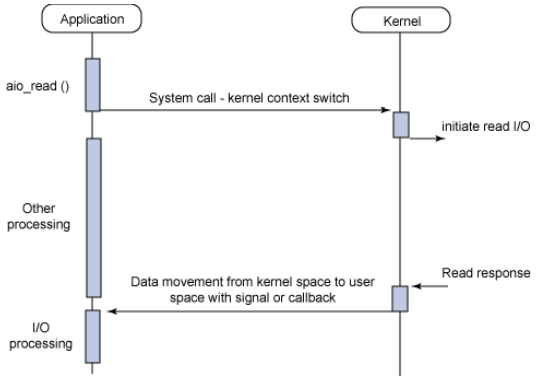
Resumen de los modos básicos de E/S en Linux

	Blocking	Non-blocking
Synchronous	Read/write	Read/write (O_NONBLOCK)
Asynchronous	I/O multiplexing (select/poll)	AIO

http://www.ibm.com/developerworks/linux/library/l-async/

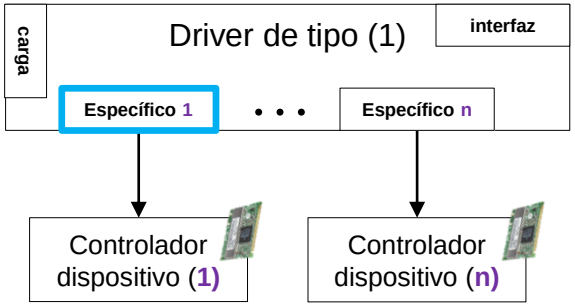
Interfaz para llamadas al sistema

Resumen de los modos básicos de E/S en Linux

	Bloqueante (notificación)	NO-bloqueante (notificación)
Síncrono (petición)		
Asíncrono (petición)		

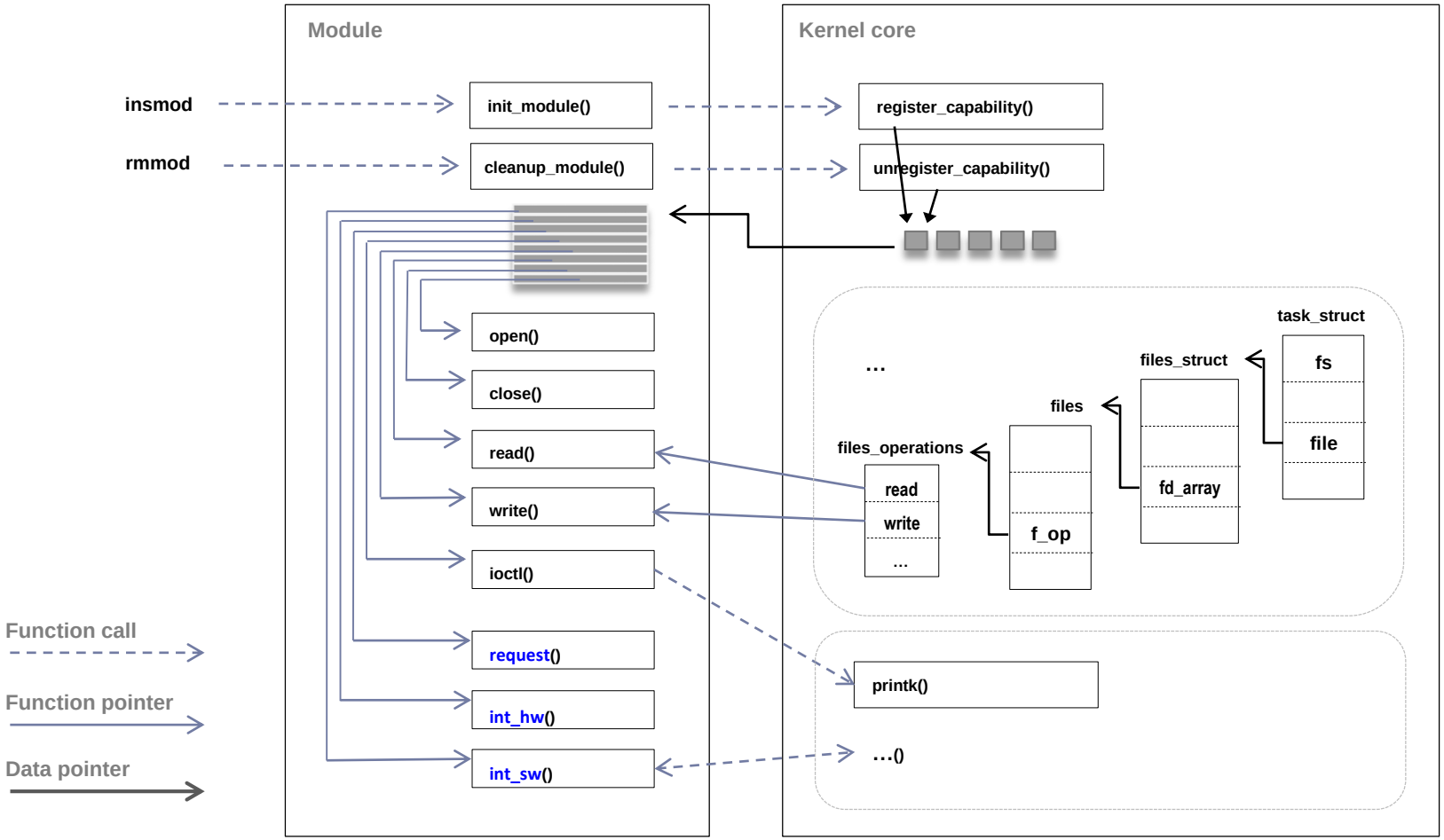
http://www.ibm.com/developerworks/linux/library/l-async/

Organización básica
3. petición al controlador de dispositivo



- Requiere implementar hasta dos funciones:
  - Función para solicitar la operación
    - Solicitada por un servicio del S.O.
  - Función para manejar la interrupción del dispositivo (al finalizar la operación)
    - Se ejecuta al recibir la interrupción.
- Necesario adaptar al tipo de controlador hardware del dispositivo:
  - Dispositivos rápidos o dispositivos *real-time*
  - D. con peticiones dependientes o independientes
  - Dispositivos proactivos o reactivos

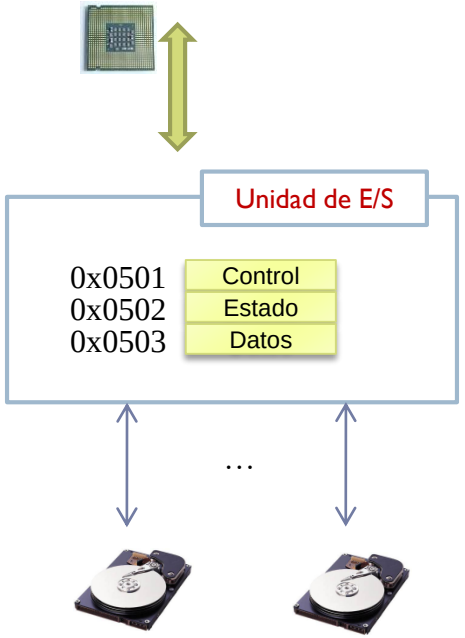
Estructuras principales de gestión
Linux



http://free-electrons.com/doc/books/ldd3.pdf

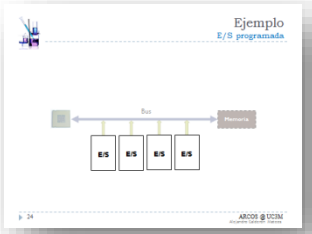
Módulo de E/S
resumen de las características fundamentales

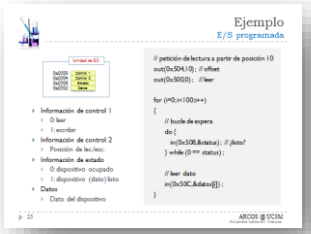
- Unidad de transferencia
  - De bloques
  - De caracteres
- Direccionamiento
  - Proyectados en memoria
  - Mediante puertos
- Interacción computador-controlador
  - E/S programada o directa
  - E/S por interrupciones
  - E/S por DMA
- Tipos principales de protocolos
  - Petición-respuesta individual
  - Petición-respuesta compartida
  - Solo petición
  - Solo respuesta

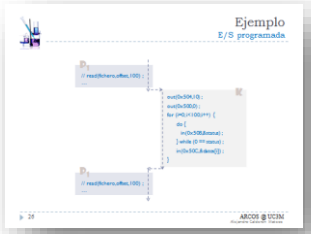


Interacción computador-controlador

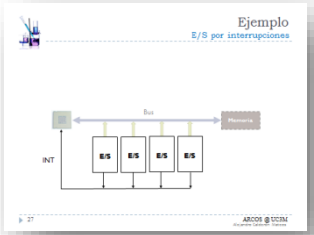
- **E/S programada o directa**

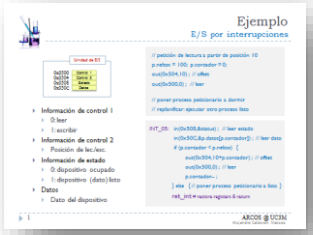


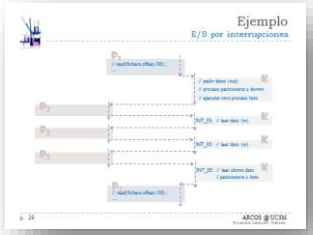




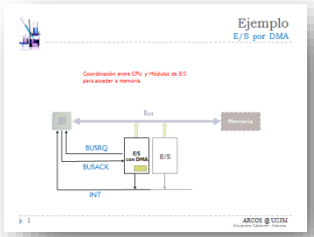
- **E/S por interrupciones**

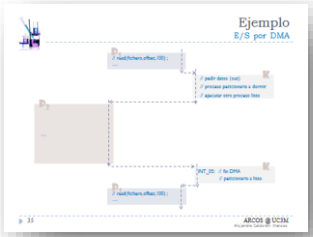




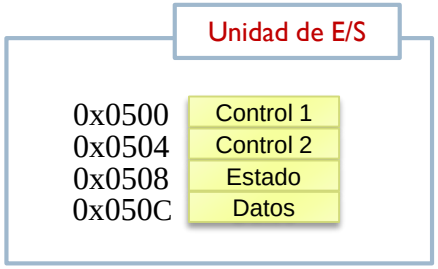


- **E/S por DMA**





Ejemplo
E/S programada

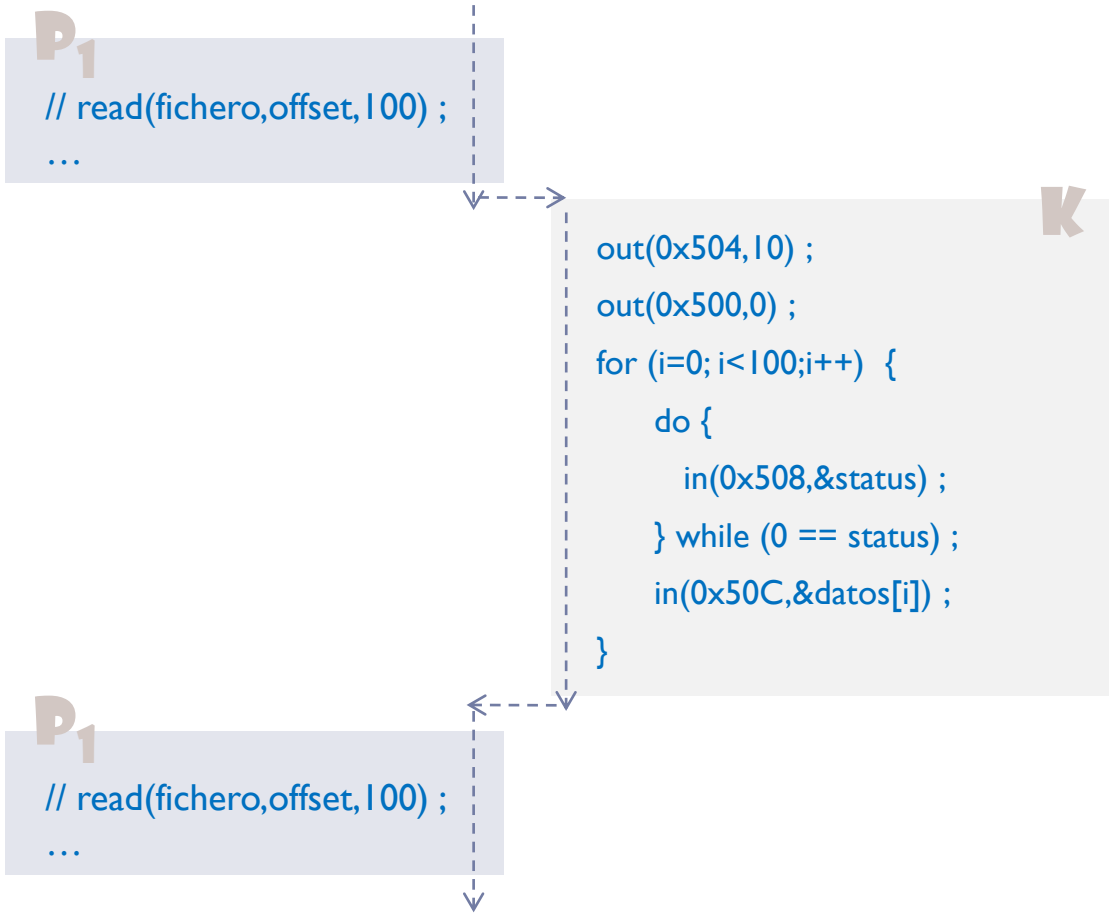


- Información de control 1
  - 0: leer
  - 1: escribir
- Información de control 2
  - Posición de lec./esc.
- Información de estado
  - 0: dispositivo ocupado
  - 1: dispositivo (dato) listo
- Datos
  - Dato del dispositivo

petición:

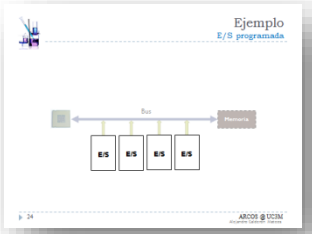
```
for (i=0; i<100;i++)
{
      // pedir leer a partir de la posición 10
      out(0x504,10) ;
      out(0x500,0) ;

      // bucle de espera
      do {
        in(0x508,&status) ;  // ¿listo?
      } while (0 == status) ;

      // leer dato
      in(0x50C,&(datos[i])) ;
}
```

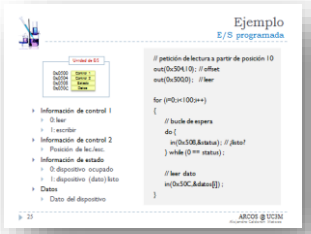
Ejemplo
E/S programada

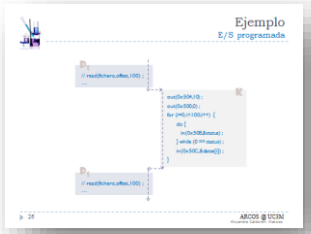


Interacción computador-controlador

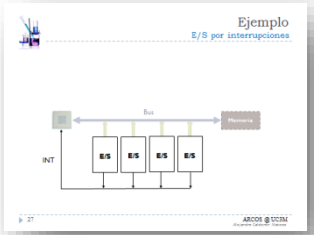
- **E/S programada o directa**

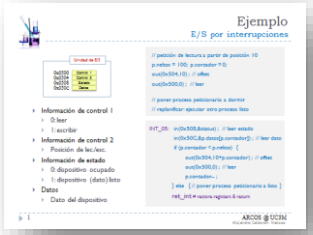


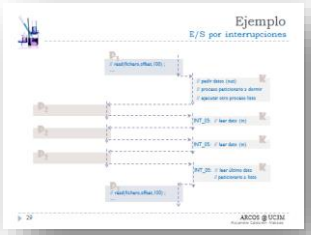




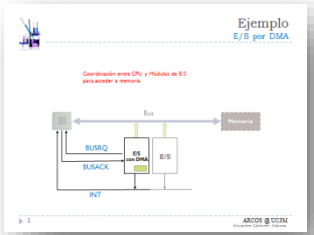
- **E/S por interrupciones**

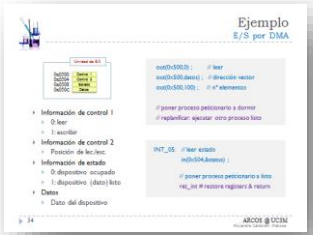


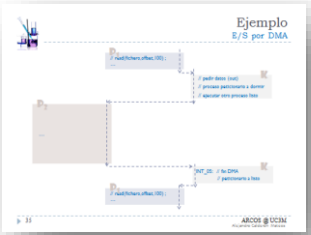




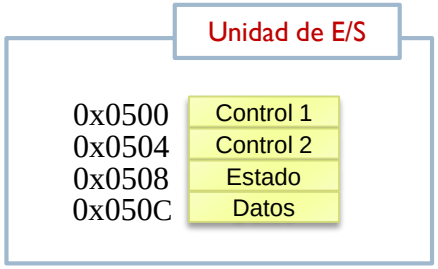
- **E/S por DMA**







Ejemplo
E/S por interrupciones

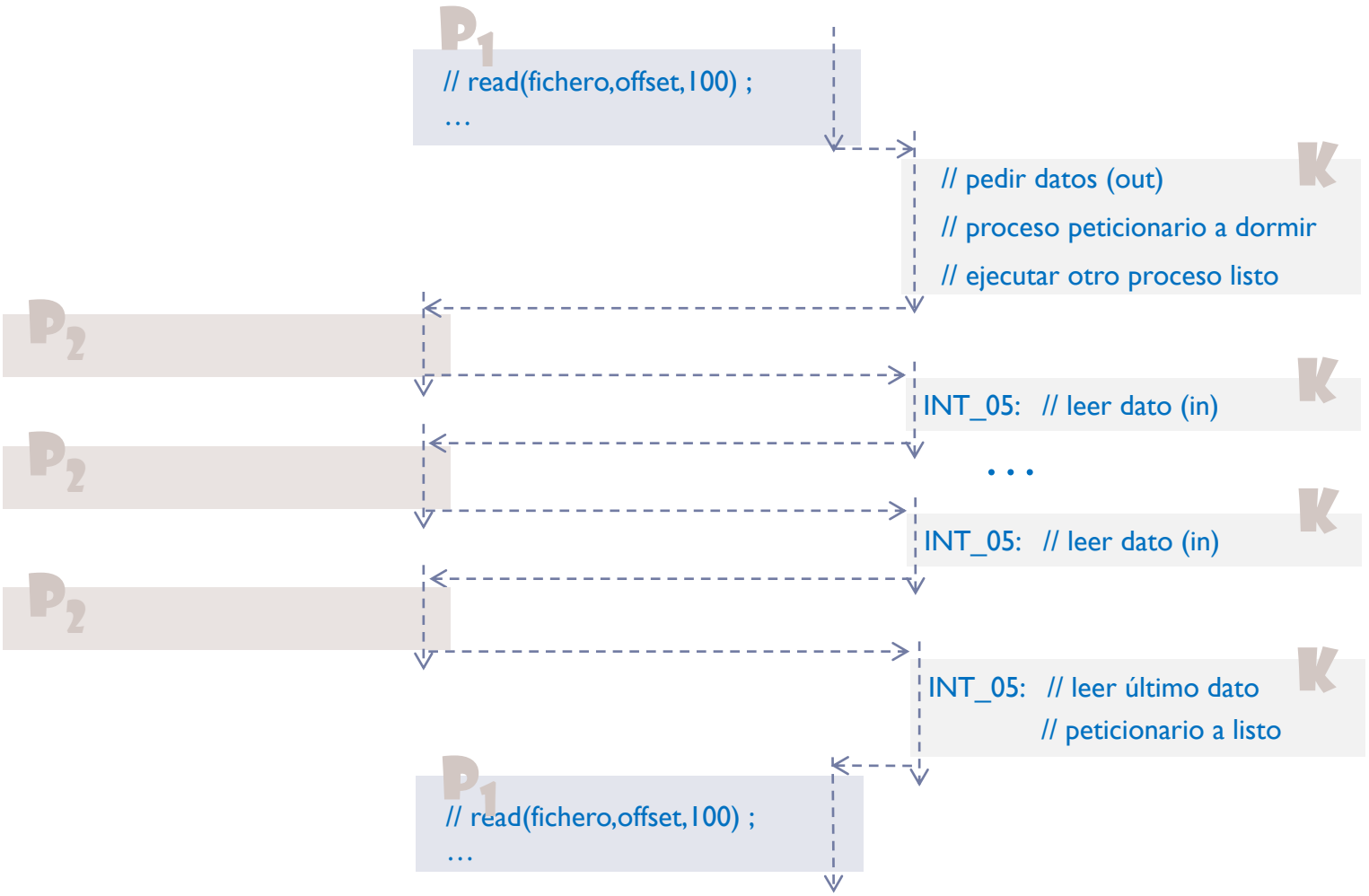


- Información de control 1
  - 0: leer
  - 1: escribir
- Información de control 2
  - Posición de lec./esc.
- Información de estado
  - 0: dispositivo ocupado
  - 1: dispositivo (dato) listo
- Datos
  - Dato del dispositivo

```
petición:
// lectura a partir de posición 10
   p.neltos = 100;  p.contador = 0;
   out(0x504,10) ;  // offset
   out(0x500,0) ;   // leer
// cambio de contexto voluntario (C.C.V.)
```

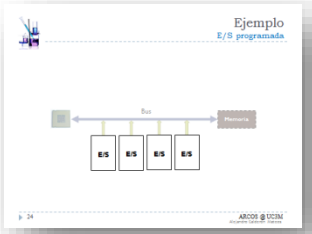
```
INT_05:  in(0x508,&status) ;   // leer estado
         in(0x50C,&p.datos[p.contador]) ;  // leer dato
         if (p.contador < p.neltos)  {
               p.contador++ ;
               out(0x504,10+p.contador) ;  // offset
               out(0x500,0) ;  // leer
         } else  { // poner proceso peticionario a listo }
         ret_int # restore registers & return
```

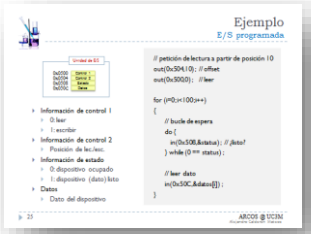
Ejemplo
E/S por interrupciones

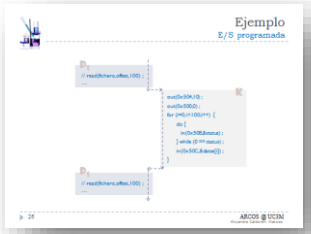


Interacción computador-controlador

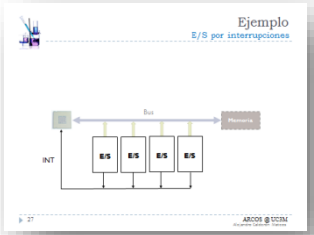
- **E/S programada o directa**

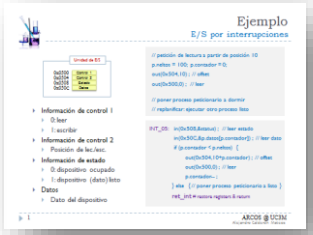


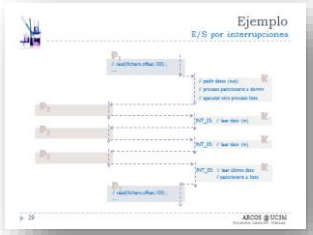




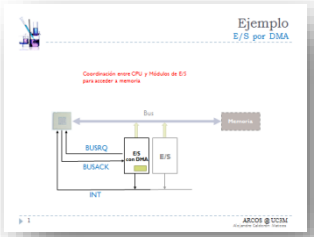
- **E/S por interrupciones**

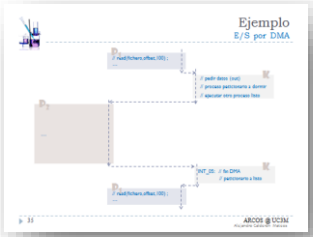




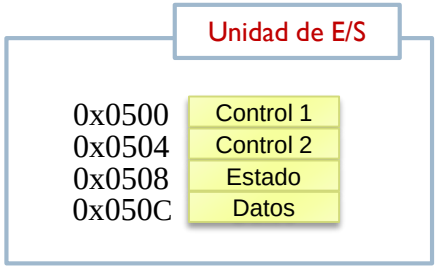


- **E/S por DMA**





Ejemplo
E/S por DMA

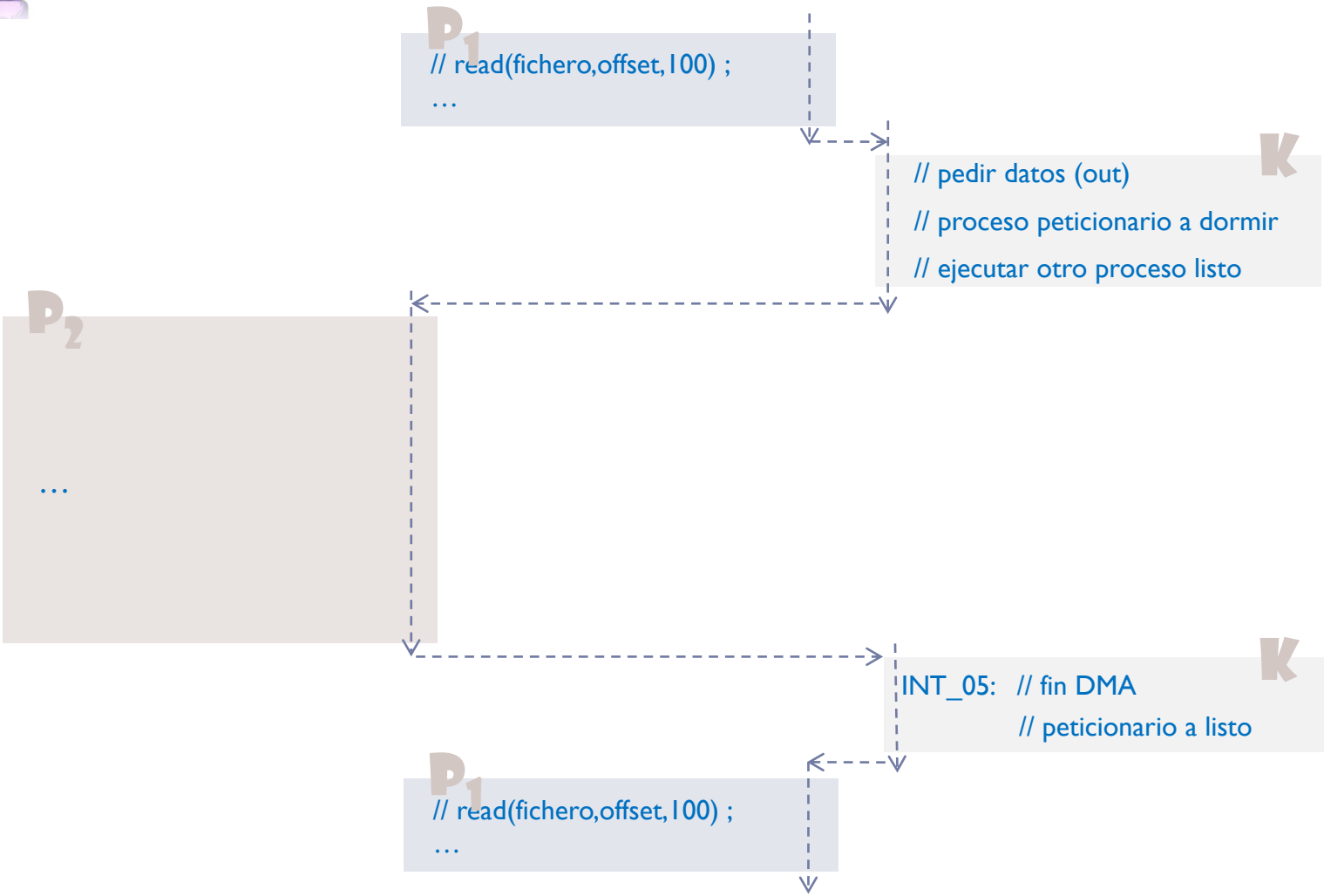


- Información de control 1
  - 0: leer
  - 1: escribir
- Información de control 2
  - Posición de lec./esc.
- Información de estado
  - 0: dispositivo ocupado
  - 1: dispositivo (dato) listo
- Datos
  - Dato del dispositivo

```
petición:
// Programar la petición del bloque
   out(0x500,0) ;        // leer
   out(0x500,datos) ;    // dirección vector
   out(0x500,100) ;      // nº elementos
   out(0x504,10) ;       // posición lec./esc.
// Cambio de contexto voluntario (C.C.V.)
```

```
INT_05:   // leer estado y datos
          in(0x508,  &status) ;
          in(0x50C, &status) ;

          // poner proceso peticionario a listo
          ret_int # restore registers & return
```

Ejemplo
E/S por DMA



Comunicación con el controlador

Linux

dso/test3.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/workqueue.h>
#include <linux/interrupt.h>
#include <linux/slab.h>

MODULE_LICENSE("Dual BSD/GPL");
MODULE_DESCRIPTION("DSO Device Driver Demo");

struct wq1_work
{
        struct work_struct work;
             unsigned char status;
                          char scancode;
};

static struct workqueue_struct *wq1 = 0;
```

Comunicación con el controlador

Linux

dso/test3.c

```
static void got_char (struct work_struct *work) {
        struct wq1_work *_w;

        _w = container_of (work, struct wq1_work, work);
        printk (KERN_INFO ” test3: scan Code %x %s.\n”,
              (int)(_w->scancode) & 0x7F, (_w->scancode) & 0x80 ? "Released" : "Pressed");
        kfree (_w);
}

irqreturn_t irq_handler (int irq, void *dev_id) {
        struct wq1_work *task;

        task = kmalloc (sizeof(struct wq1_work), GFP_KERNEL);
        task->status    = inb (0x64);
        task->scancode = inb (0x60);
        INIT_WORK (&(task->work),
                          got_char);
        queue_work (wq1, &(task->work));
        return IRQ_HANDLED;
}
```

Old kernels

```
static int initialised = 0;
…
if (initialised == 0)
        { INIT_WORK          (&(task.work), got_char); }
else { PREPARE_WORK (&(task.work), got_char); }
initialised = 1;
```

Comunicación con el controlador

Linux

dso/test3.c

```
int test3_init (void) {
    printk (KERN_INFO " test3: inserting the new irq-hander…\n");
    wq1 = create_singlethread_workqueue ("WQsched.c");
    return request_irq (1,
                        irq_handler,
                        IRQF_SHARED,
                        "test3",
                        (void *)(irq_handler));
}

void test3_exit (void) {
    printk (KERN_INFO " test3: removing the new irq-hander…\n");
    free_irq (1, (void *)(irq_handler));
    flush_workqueue     (wq1) ;
    destroy_workqueue  (wq1) ;
}

module_init  (test3_init);
module_exit (test3_exit);
```

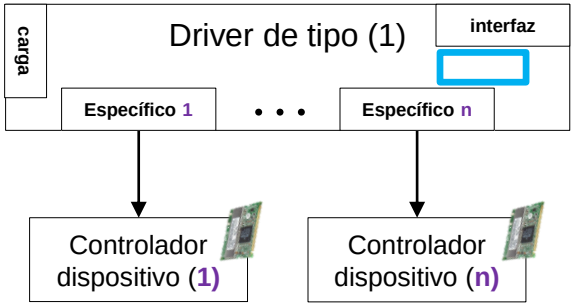
Comunicación con el controlador
Linux

dso/Makefile

```
obj-m := test1.o test2.o test3.o
```

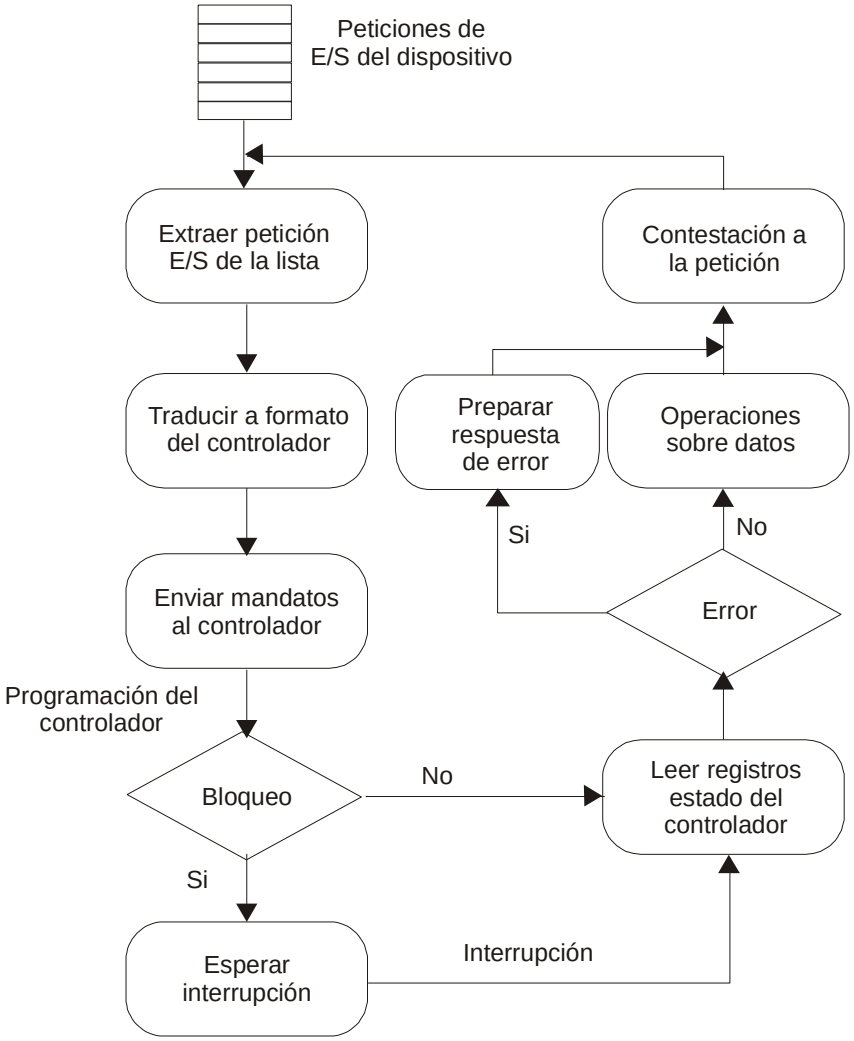
```
make -C /usr/src/linux M=`pwd` modules
tail –f /var/log/syslog &
insmod test3.ko
ls
rmmod test3
```

Organización básica
4. planificación de E/S en el driver

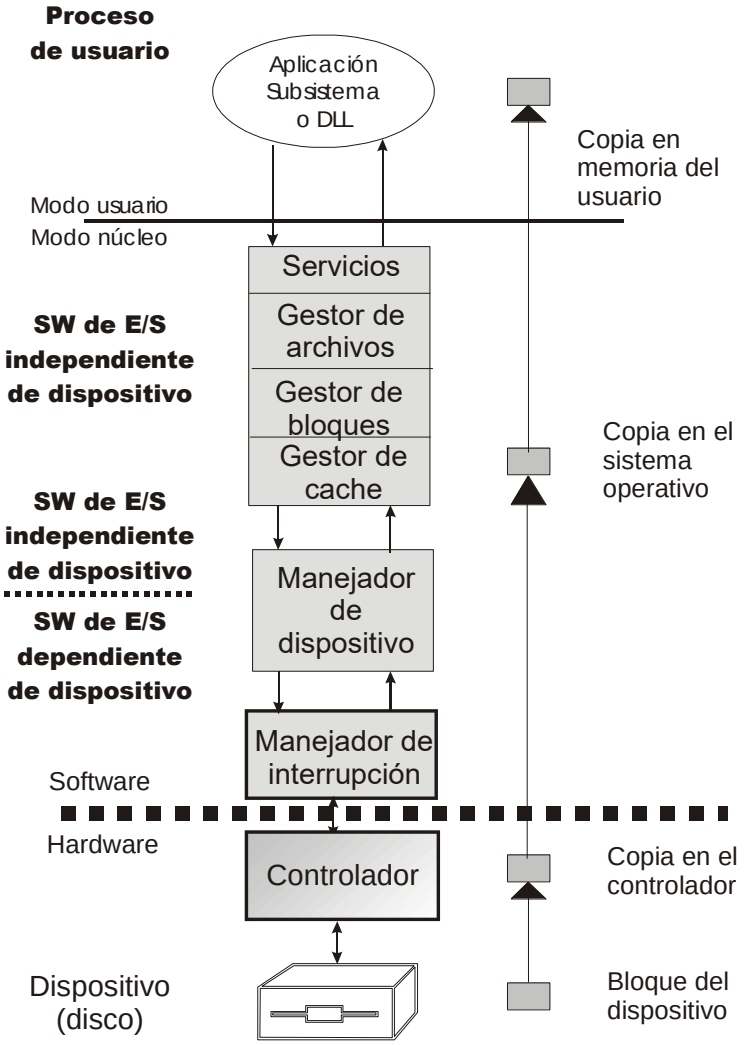


- Cuando hay varias peticiones a un dispositivo, estas se mantienen en una cola de peticiones.
  El driver suele disponer de un planificador de E/S que permite planificar las peticiones de manera que se minimice el tiempo de atención a las mismas
  - Los bloques de disco se planifican para minimizar el tiempo gastado en mover las cabezas del disco.
- El planificador de E/S suele realizar, al menos, dos operaciones básicas:
  - Ordenación: las peticiones se insertan en una lista según algún criterio
  - Fusión: dos peticiones pequeñas consecutivas se transforman en una única petición.

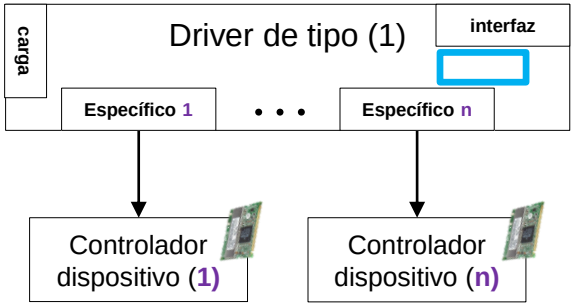
Planificación de E/S en el driver



Flujo de una operación de E/S

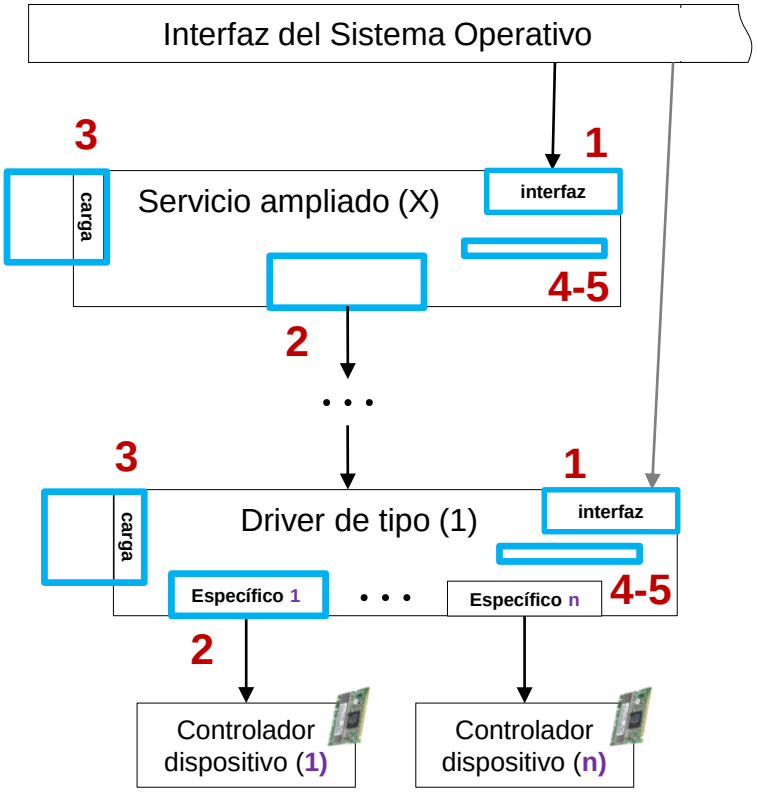


Organización básica
5. inicialización y finalización del driver



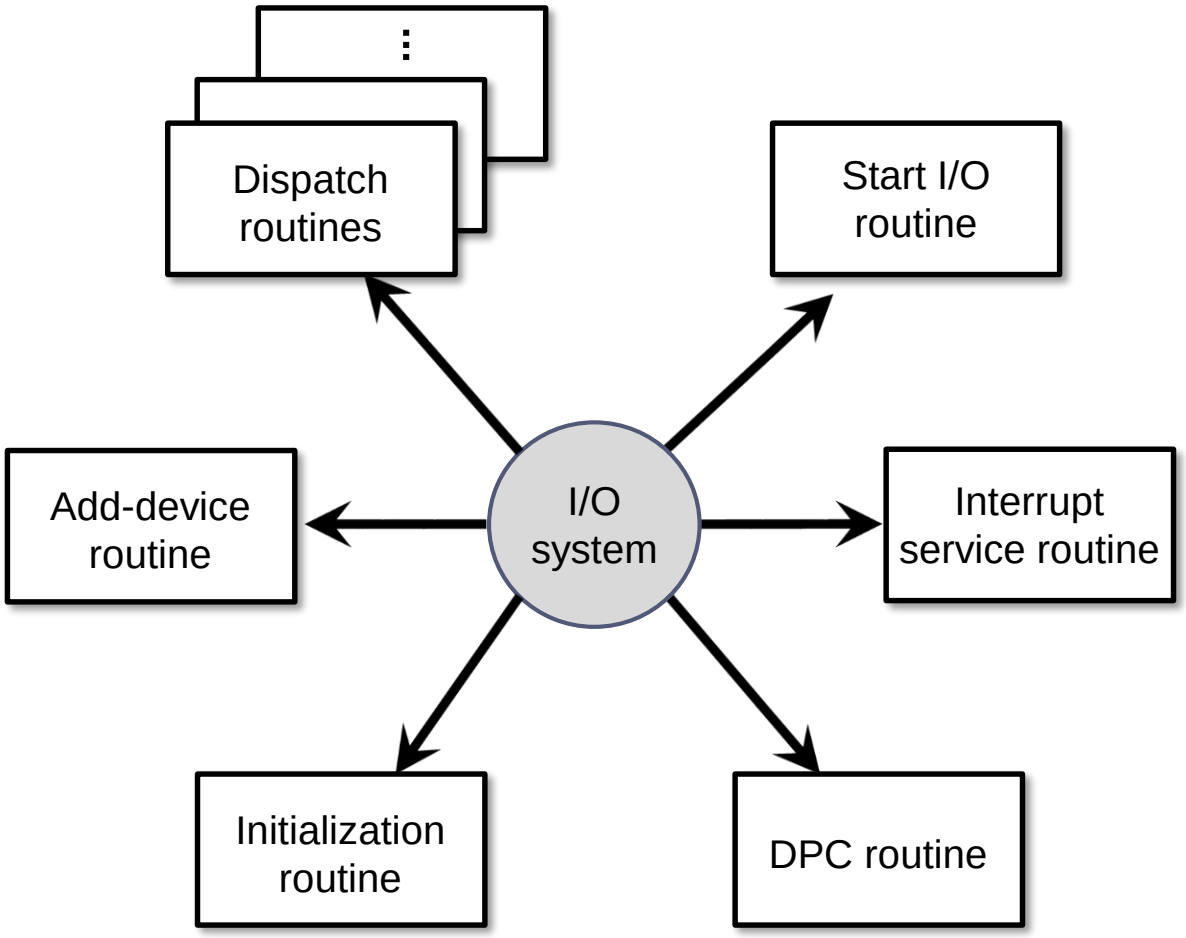
- Cuando se está utilizando un driver necesita una serie de recursos asociados (IRQ, buffer de memoria, etc.)
- Para controlar la asignación de recursos se puede seguir el siguiente esquema:
  - Un contador mantiene el número de procesos que van a trabajar con un dispositivo.
  - Cada vez que un nuevo proceso opera con un dispositivo se incrementa el contador, y cuando deja de operar se decrementa.
  - Cuando el contador pasa a 1 se realiza la asignación de recursos al driver.
  - Cuando el contador pasa a 0 se libera todos los recursos.

Organización básica
resumen

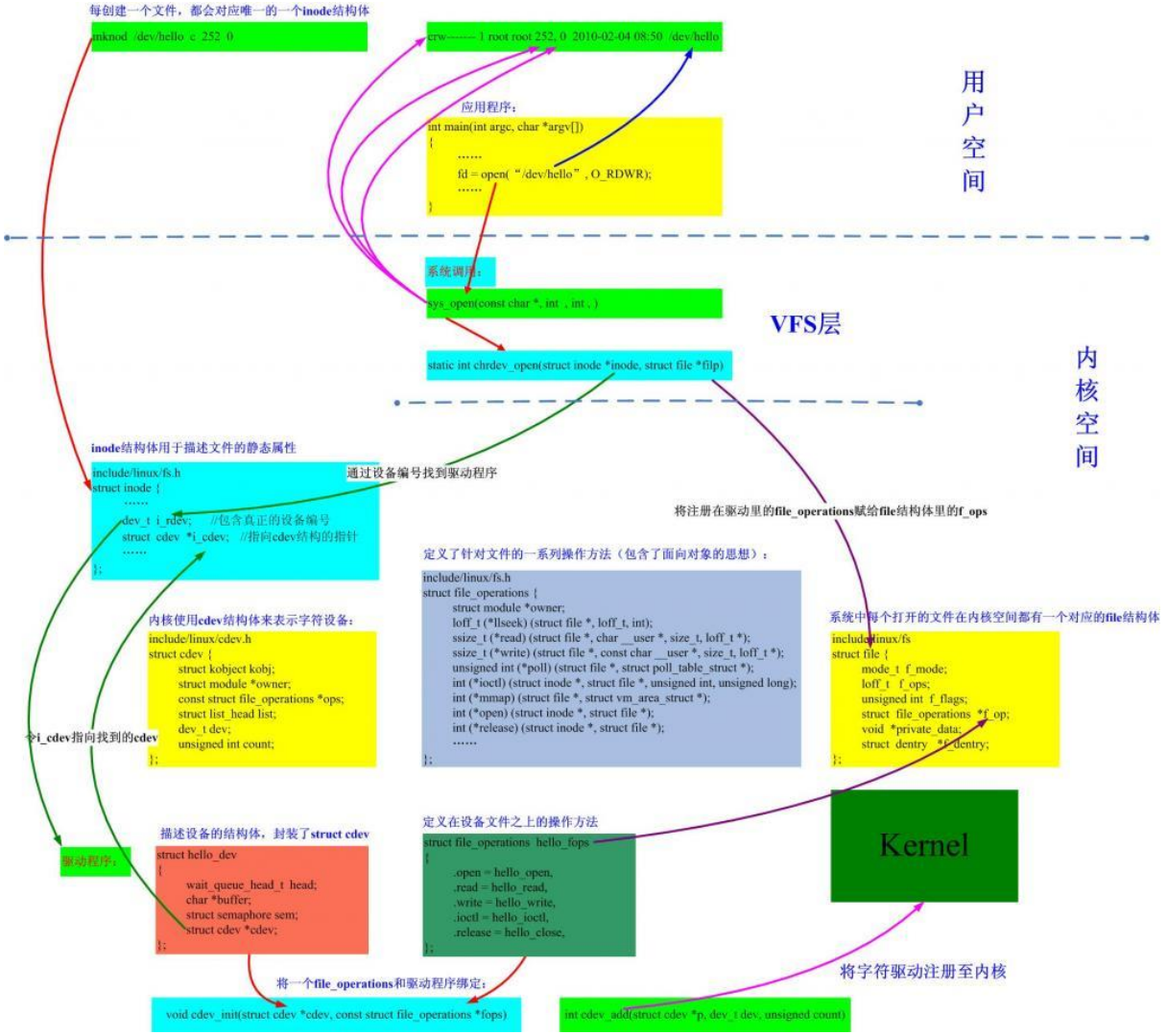


1. Interfaz para llamadas al sistema
2. Petición al controlador de dispositivo
3. Registro de drivers
4. Planificación de E/S en el driver
5. Inicialización y finalización del driver

Windows 2000
rutinas en un driver

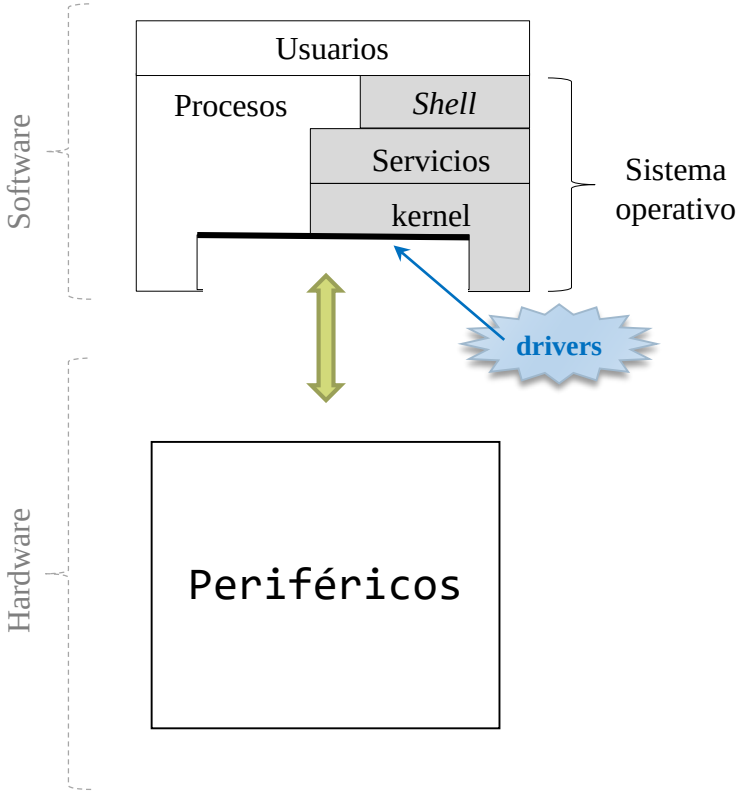


Estructuras de datos
Linux



http://blog.csdn.net/vhghhd/article/details/6853550

Contenidos



- Introducción
- Organización de los drivers
- Estructura de un driver
- **Ejemplos de diseño**

Organización básica

Ejemplos con distintos tipos de dispositivos

- Dispositivo rápido [no c.c.v]
  - Solo petición
  - Solo respuesta
- Dispositivo lento [posible c.c.v]
  - Peticiones independientes y consumible
  - Peticiones compartidas

Organización básica

Ejemplos con distintos tipos de dispositivos

- Dispositivo rápido [no c.c.v]
  - Solo petición
  - Solo respuesta

- Dispositivo lento [posible c.c.v]
  - Peticiones independientes y consumible
  - Peticiones compartidas

Petición al controlador de dispositivo
rápido (salida)

Pantalla

- Dispositivo rápido

- Petición de datos:
  - Escribir los datos en un buffer

- Manejador de interrupción del dispositivo:

Petición al controlador de dispositivo
rápido (entrada)

Reloj

- Dispositivo rápido

- Petición de datos:

- Manejador de interrupción del dispositivo:
  - Ticks = Ticks + 1

Organización básica

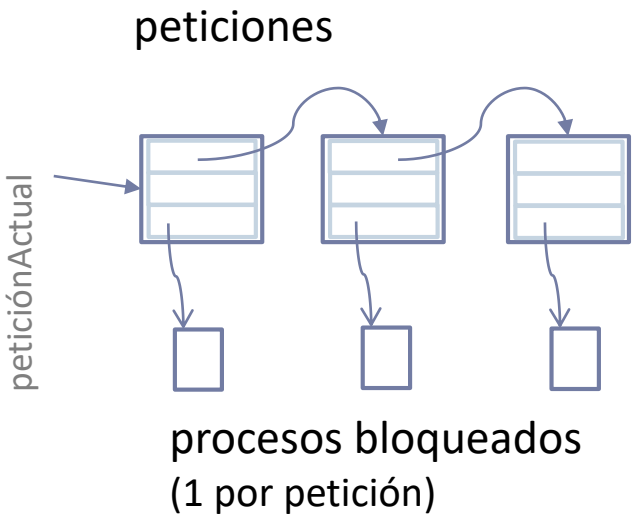
Ejemplos con distintos tipos de dispositivos

- Dispositivo rápido no c.c.v
  - Solo petición
  - Solo respuesta
- Dispositivo lento posible c.c.v
  - Peticiones independientes y consumible
  - Peticiones compartidas

Petición al controlador de dispositivo
lento / independiente (salida)

Impresora

- Dispositivo lento
- Peticiones independientes

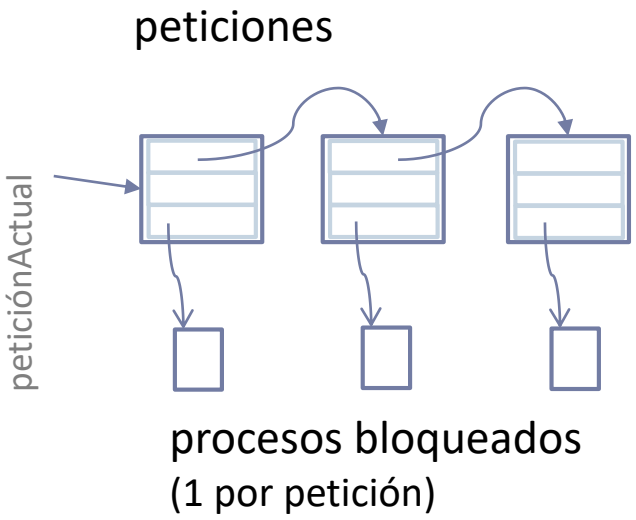


- Petición de datos:
  - Crear una petición
  - Copiar datos a petición->buffer_intermedio
  - Si no imprimiendo datos
    - Poner a imprimir petición
  - Bloquear + ejecutar otro proceso
- Manejador de interrupción del dispositivo:
  - Poner listo el proceso
  - Si hay datos a imprimir
    - Poner a imprimir siguiente petición

Petición al controlador de dispositivo
lento / independiente (entrada)

Scanner

- Dispositivo lento
- Peticiones independientes

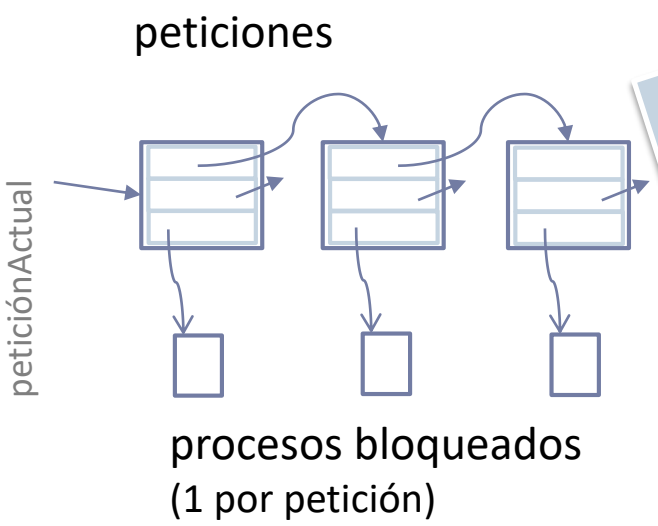


- Petición de datos:
  - Crear una petición
  - Si scanner libre
    - Poner a escanear petición
  - Bloquear + Ejecutar otro proceso
  - Copiar al usuario del buffer intermedio
- Manejador de interrupción del dispositivo:
  - Insertar datos en buffer intermedio
  - Poner listo el proceso
  - Si hay más peticiones
    - Poner a escanear siguiente petición

Petición al controlador de dispositivo
lento / independiente (entrada)

Scanner

- Dispositivo lento
- Peticiones independientes



- Petición de datos:
  - Crear una petición
  - Si scanner libre
    - Poner a escanear petición
  - Bloquear + Ejecutar otro proceso
  - ~~Copiar al usuario del buffer intermedio~~
- Manejador de interrupción del dispositivo:
  - Copiar datos al buffer del usuario
  - Poner listo el proceso
  - Si hay más peticiones
    - Poner a escanear siguiente petición

Organización básica

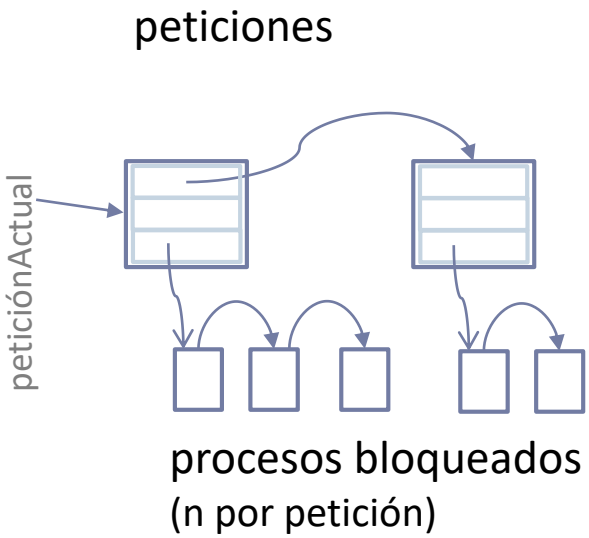
Ejemplos con distintos tipos de dispositivos

- Dispositivo rápido [no c.c.v]
  - Solo petición
  - Solo respuesta
- Dispositivo lento [posible c.c.v]
  - Peticiones independientes y consumible
  - Peticiones compartidas

Petición al controlador de dispositivo
lento / compartidas (salida)

Disco

- Dispositivo lento
- Peticiones compartidas

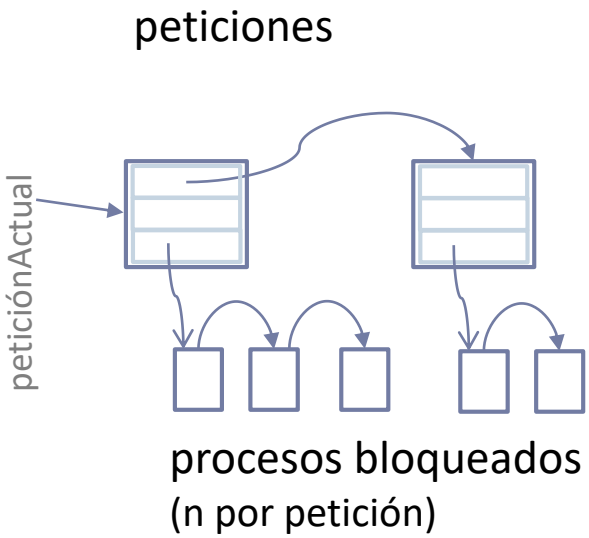


- Petición de datos:
  - Si otro proceso realizó la petición
    - Actualizar los datos
    - Dormir por esa petición
  - En caso contrario
    - Construir una nueva petición
    - Encolar la petición
    - Actualizar los datos
    - Dormir en esa petición
- Manejador de interrupción del dispositivo:
  - Despertar a todos los procesos dormidos por la petición completada
  - Si hay peticiones pendientes
    - Ir pidiendo la siguiente petición

Petición al controlador de dispositivo
lento / compartidas (entrada)

CDROM

- Dispositivo lento
- Peticiones compartidas



- Petición de datos:
  - Si otro proceso realizó la petición
    - Dormir por esa petición
  - En caso contrario
    - Construir una nueva petición
    - Encolar la petición
    - Dormir en esa petición
  - Copiar al usuario del buffer intermedio
- Manejador de interrupción del dispositivo:
  - Insertar datos en buffer intermedio
  - Despertar a todos los procesos dormidos por la petición completada
  - Si hay peticiones pendientes
    - Ir pidiendo la siguiente petición

Organización básica

Ejemplos con distintos tipos de dispositivos

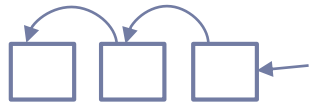
- Dispositivo rápido [no c.c.v]
  - Solo petición
  - Solo respuesta
- Dispositivo lento [posible c.c.v]
  - Peticiones independientes y consumible
  - Peticiones compartidas

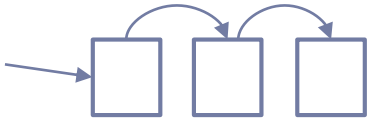
Petición al controlador de dispositivo
lento / independiente (entrada)

Teclado

- Dispositivo lento
- Peticiones independientes

- Petición de datos:
  - Si NO hay datos
    - Dormir + Ejecutar otro proceso
  - Leer los datos de un buffer





procesos bloqueados
(si no hay datos)

- Manejador de interrupción del dispositivo:
  - Insertar datos en buffer
  - Si hay procesos dormidos
    - Despertar al primero

Grupo ARCOS
Universidad Carlos III de Madrid

Lección 3c

procesos, periféricos, *drivers* y servicios ampliados

Diseño de Sistemas Operativos
Grado en Ingeniería Informática

CC BY NC SA