ARCOS Group
Computer Science and Engineering Department
Universidad Carlos III de Madrid

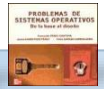
Lesson 3c

process, devices, drivers, and extended services

Operating System Design
Degree in Computer Science and Engineering, Double Degree CS&E + BA

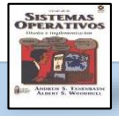
Recommended readings

Base

1. Carretero 2007:
   1. Cap.7

Recommended

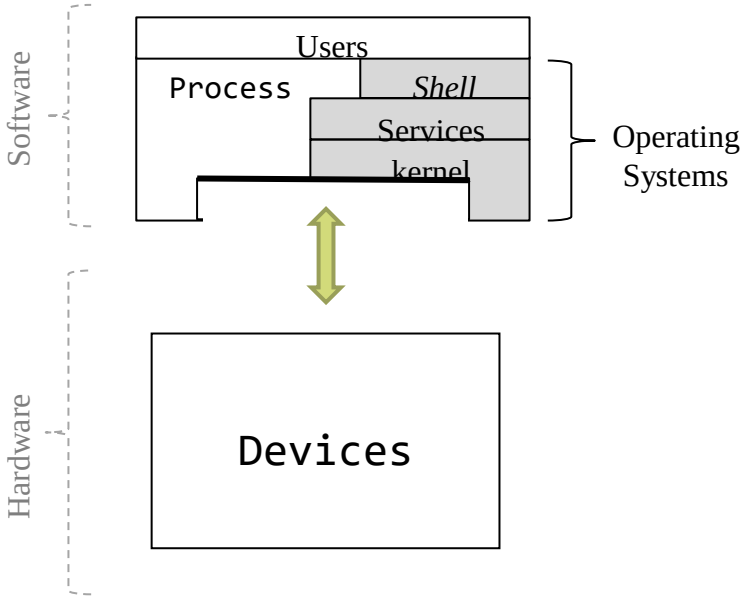


1. Tanenbaum 2006(en):
   1. Cap.3
1. Stallings 2005(en):
   1. Parte tres
1. Silberschatz 2006:
   1. Cap. Sistemas de E/S

To remember…

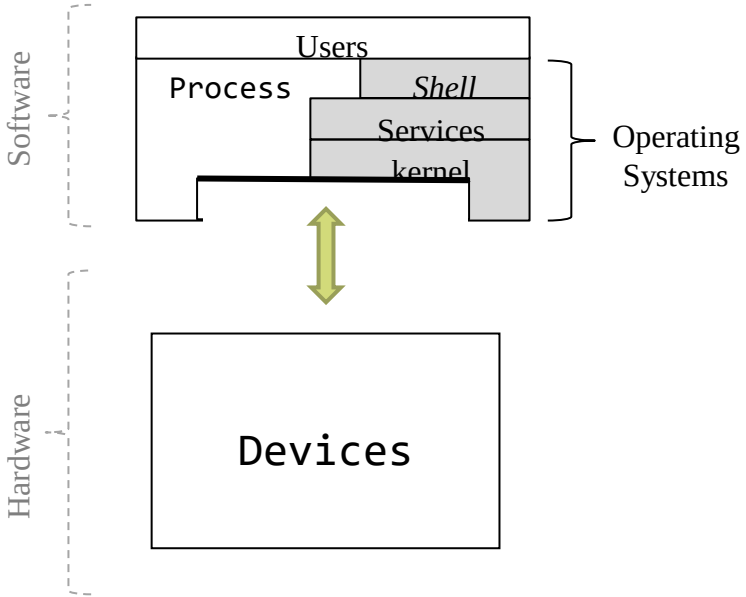
1. To prepare and review the class explanations.
   - Study the bibliography material: only slides are not enough.
   - Ask your doubts.
1. To exercise skills and abilities.
   - Solve as much exercises as possible.
   - Perform the guided laboratories progressively.
   - Build laboratories progressively.

Overview



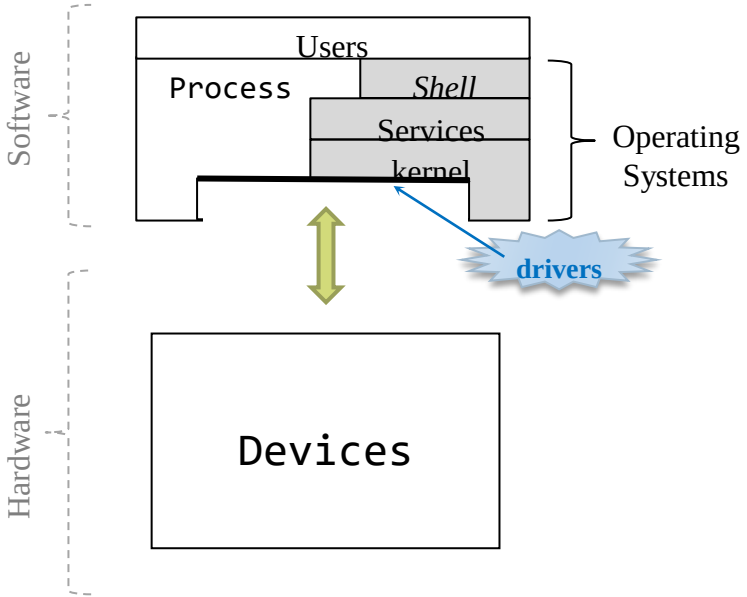
- Introduction
- Driver framework
- Structure of one driver
- Driver design examples

Overview



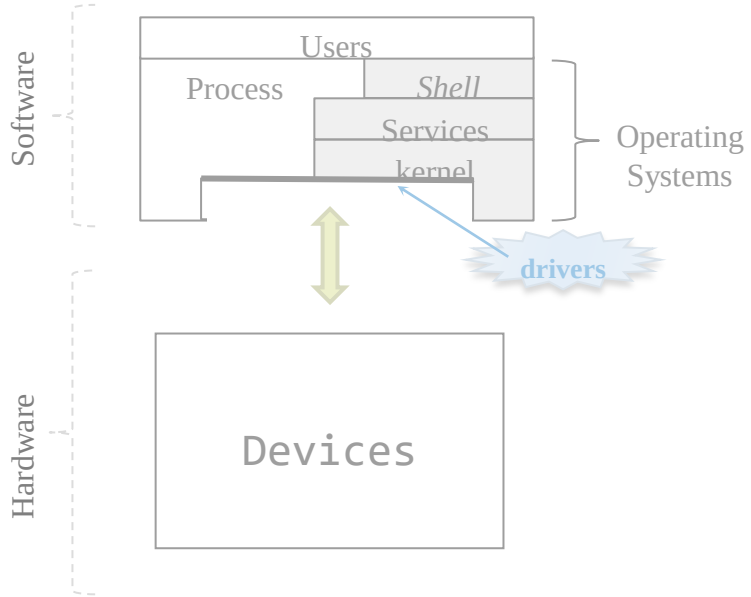
- **Introduction**
- Driver framework
- Structure of one driver
- Driver design examples

Scope of management



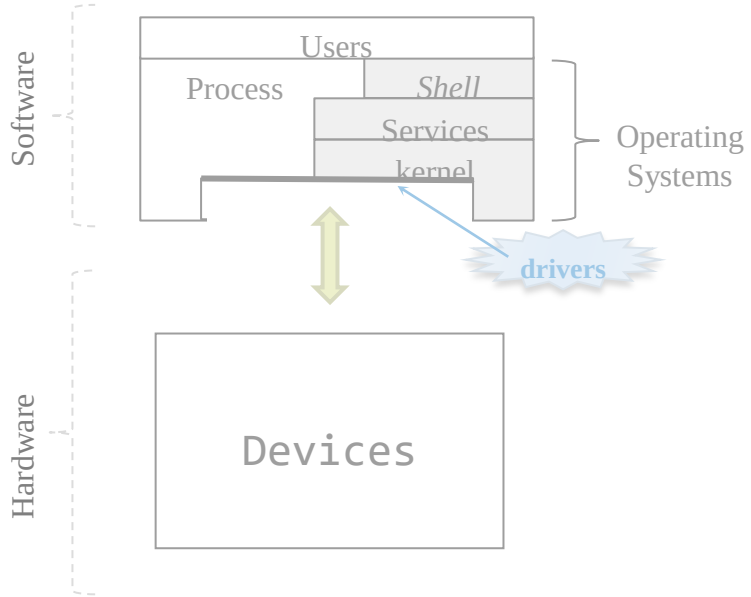
- Part of the operating system responsible for the interaction with all possible devices (hardware)
- It includes all the communication of the CPU and memory with the rest of hardware elements.

Characteristics by the management scope



- Operating System dependant:
  - The drivers for one operating system are not easy to be reused in another operating system.
- It is a very dynamic part:
  - Drivers are added continuously.
- Implemented in modules:
  - Add/remove without stop O.S.

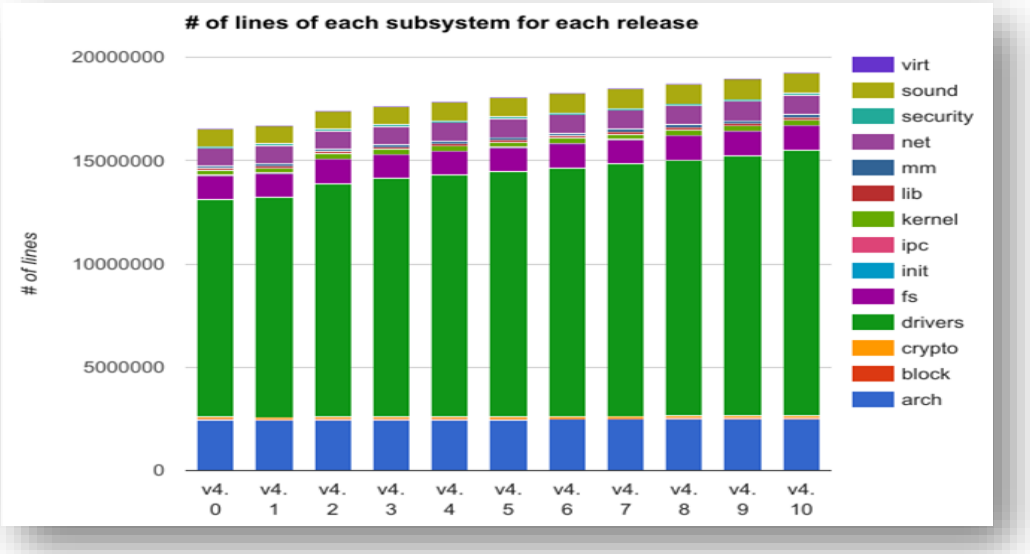
Goals of the I/O



- To offer a simplified logical vision for:
  - Rest of the operating system
  - Processes and Users
- To optimize I/O performance
- Facilitate the management of peripherals
- Facilitate adding support to new devices

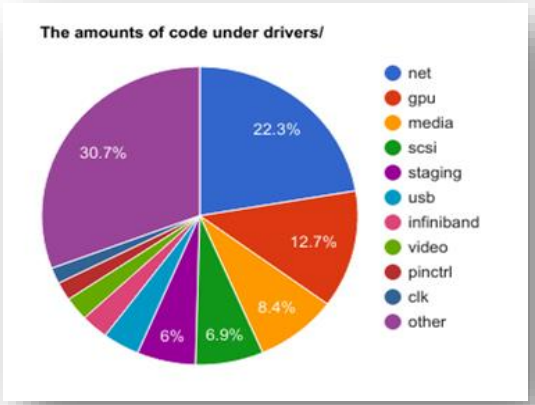
Importance of drivers

- Statistics Linux kernel 4.10:
  - ~21 millions of source code lines.
  - Most of the code are drivers:

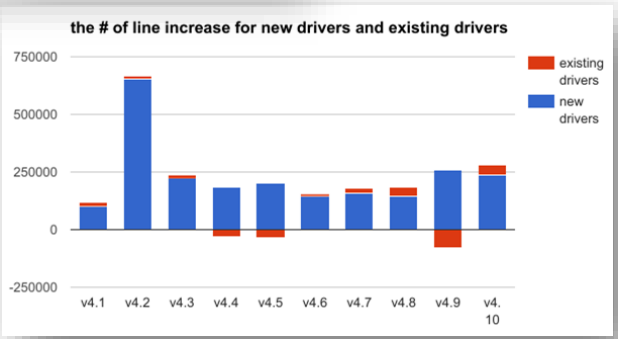


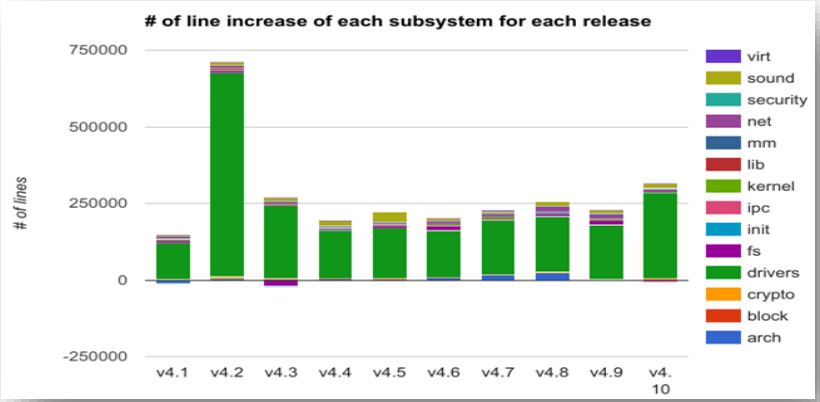
https://github.com/satoru-takeuchi/linux-kernel-statistics

Importance of drivers



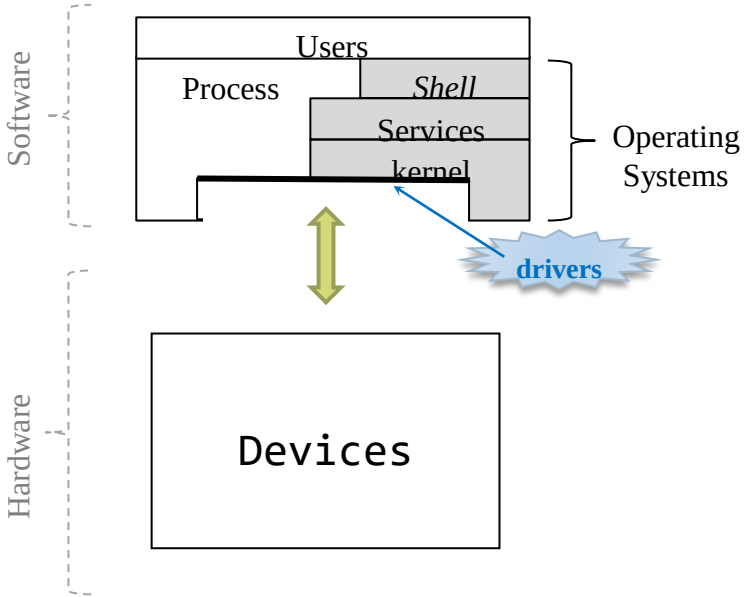
- Most added/removed code are drivers.
- It is a code that works with full access to the system (same level of protection as the kernel)





https://github.com/satoru-takeuchi/linux-kernel-statistics

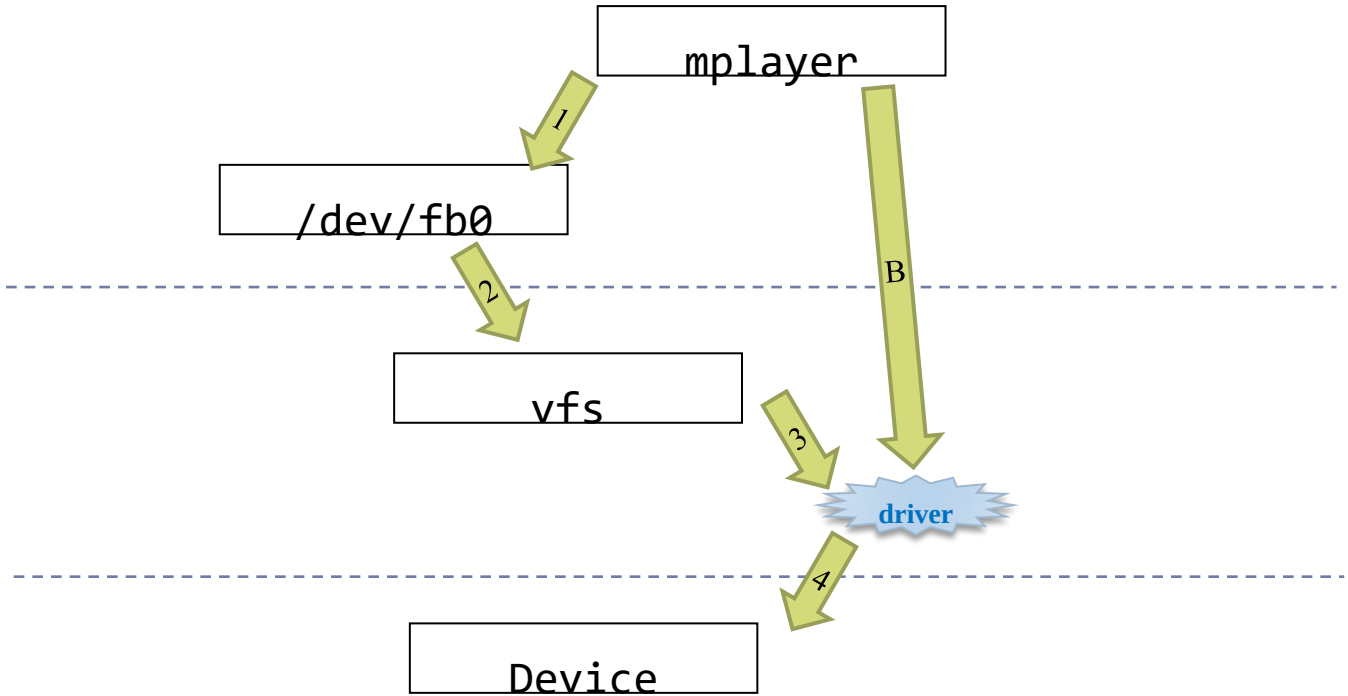
Overview



- Introduction
- **Driver framework**
- Structure of one driver
- Driver design examples

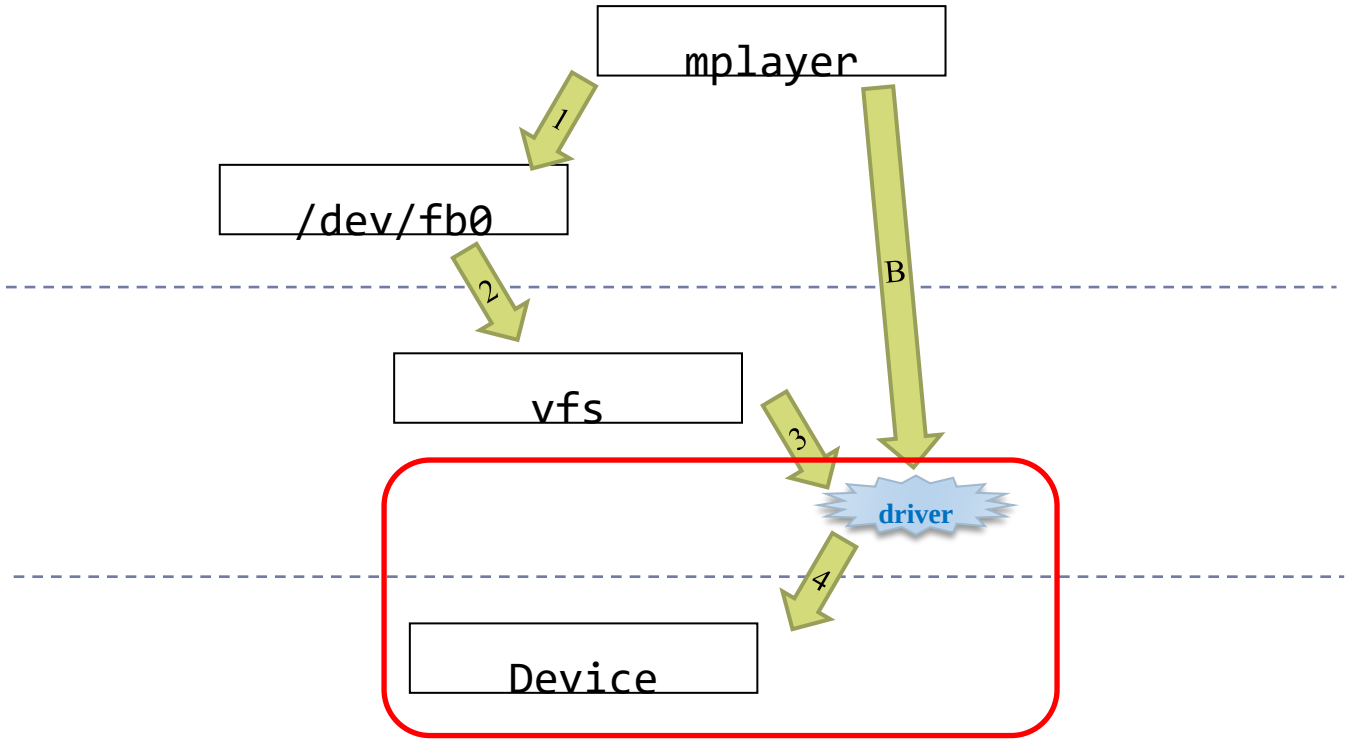
Simplified logic vision

Linux



Simplified logic vision

Linux



Hardware inventory

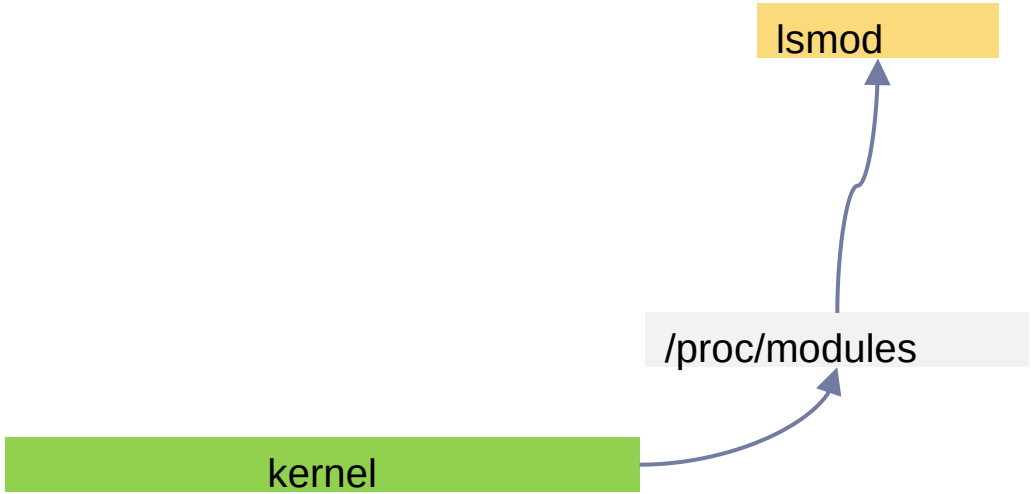
Linux

- When kernel boots:
  - it will discover the available peripherals and it associate the most appropriate driver available.
- When a hot-pluggable devices is connected (like USB devices ones)
  - it discover the peripheral added and dynamically search the most appropriate driver.

```
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
…
```

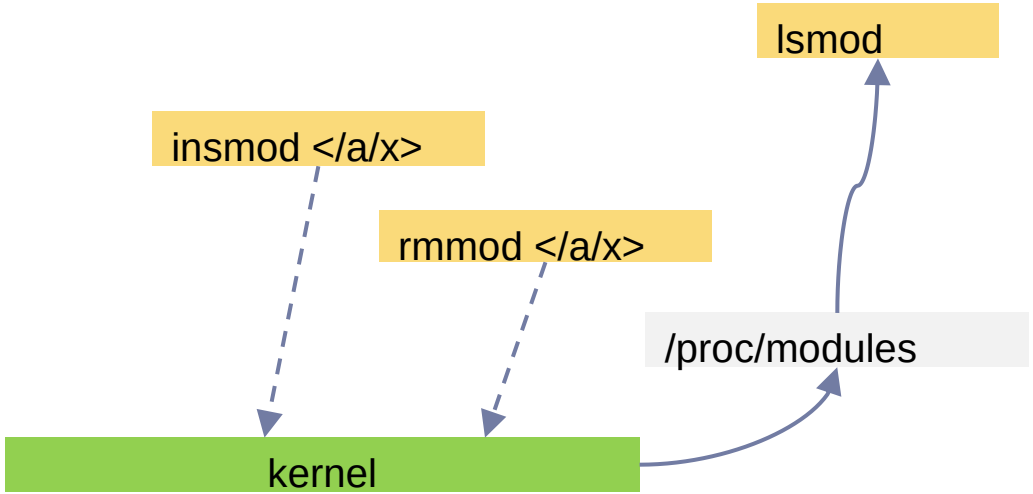
Basic drivers management

Linux: list



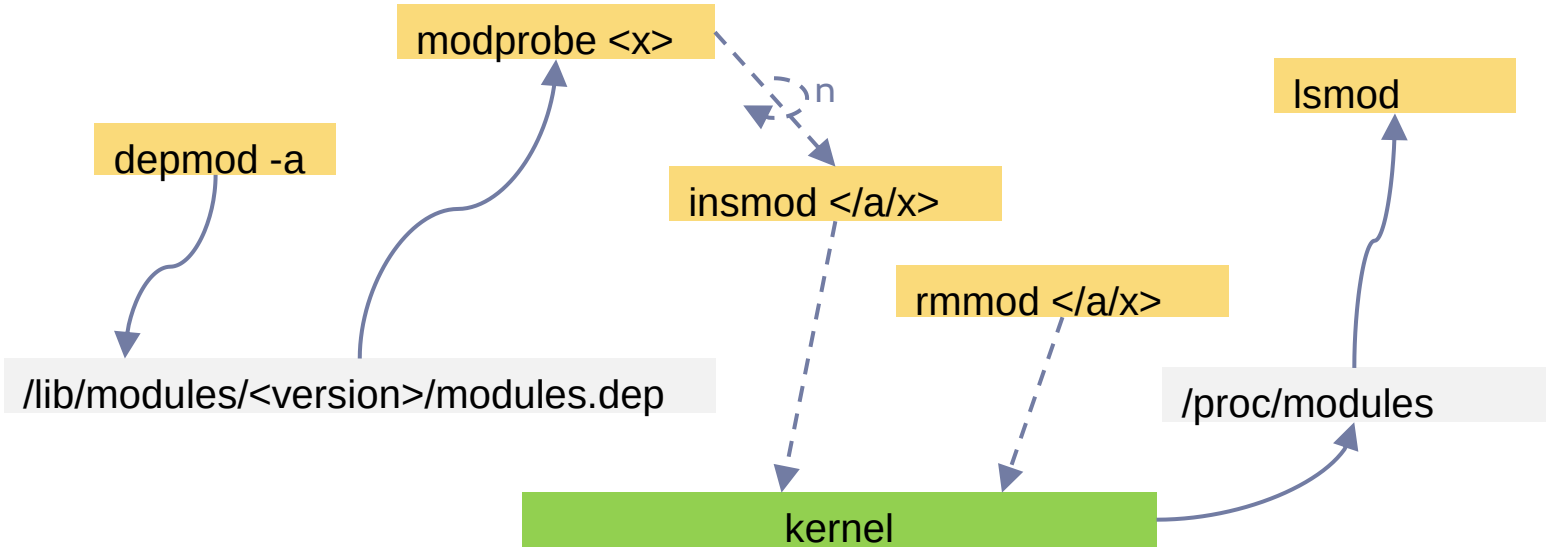
Basic drivers management

Linux: add/remove a driver manually



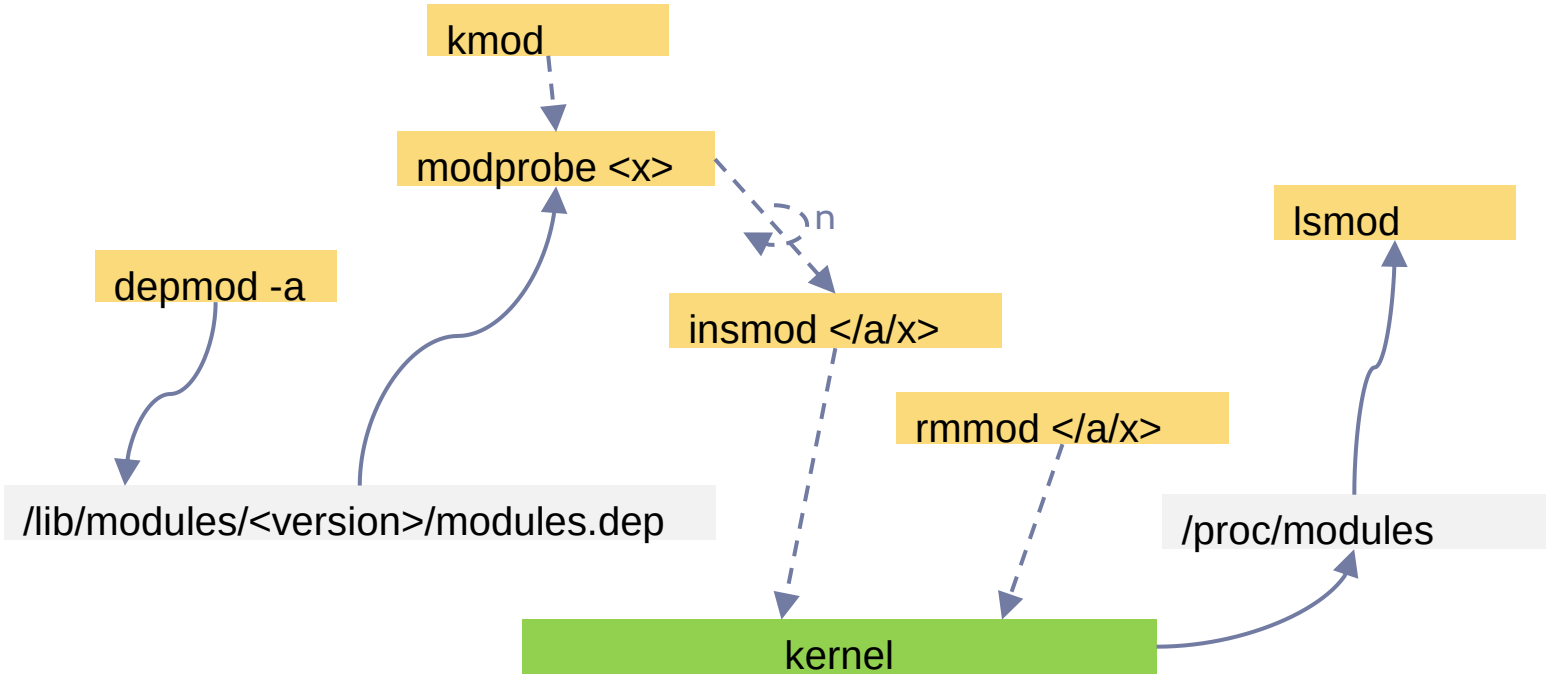
Basic drivers management

Linux: add with dependencies



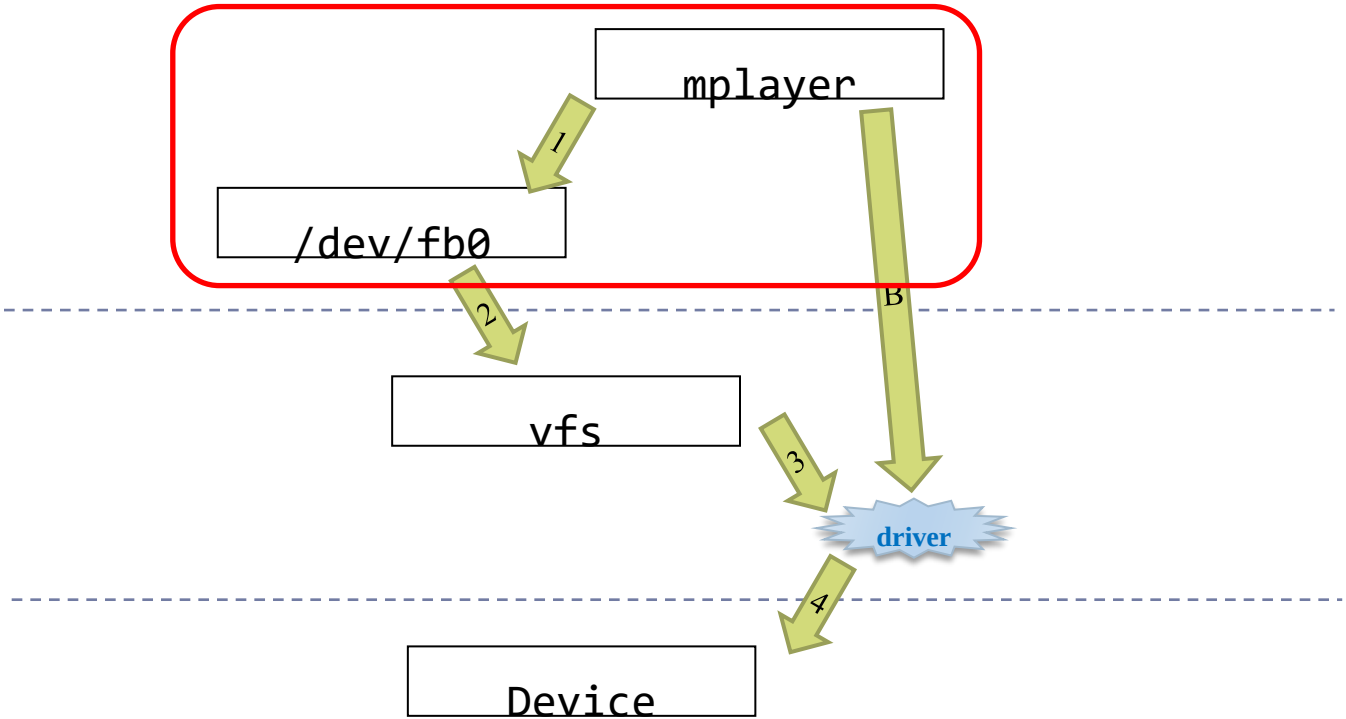
Basic drivers management

Linux



Simplified logic vision

Linux



Representation used for devices

Linux

- It is usually files at /dev: `/dev/xxxx`
  - It could be files for virtual devices: standard input-output, etc.
  - And devices without files: network cards, 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

Representation used for devices
Linux

- There are identified by:
  - *Major number* (driver) + *minor number* (“device”)

```
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

Representation used for devices

Linux

- They are managed by:
  - mkdev (deprecated): script to create all possible files
  - devfs (deprecated): file system with all possible devices
  - udev: dynamic file system (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

Representation used for devices

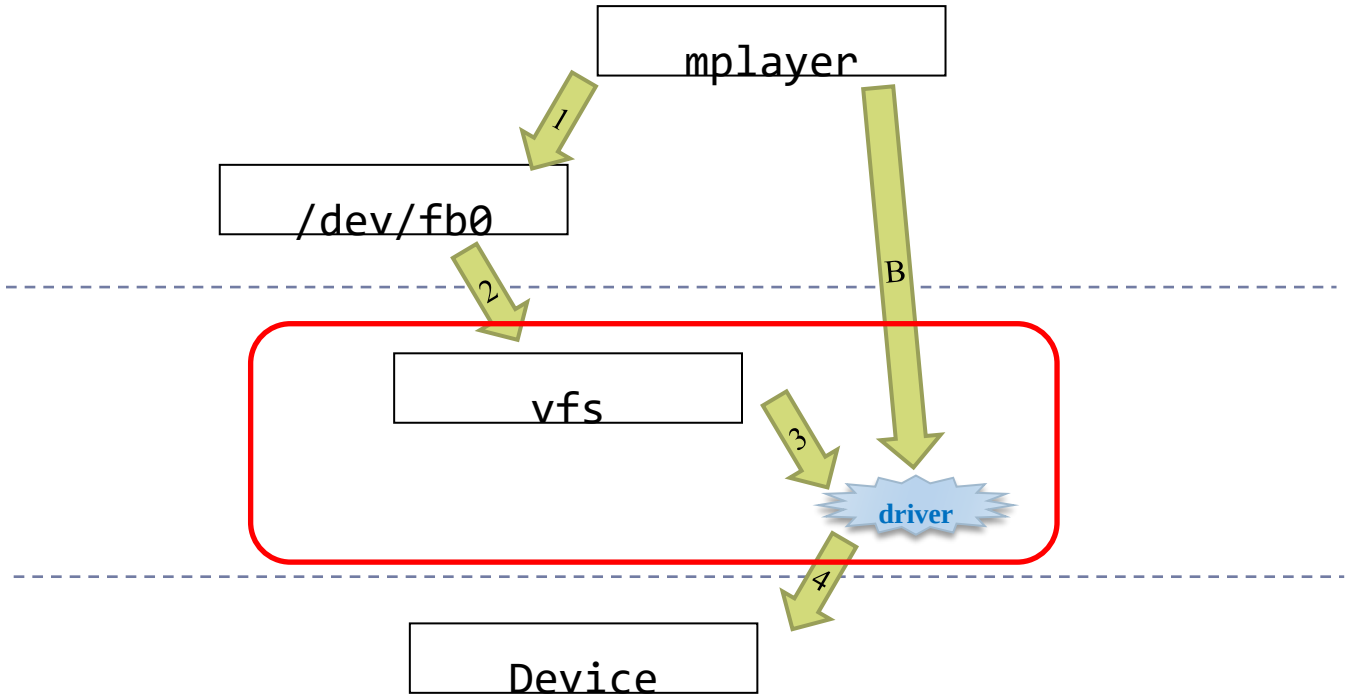
Linux

- It is possible to manually create a new device file:
  - Name of file
  - Type: block or character
  - Major & minor number

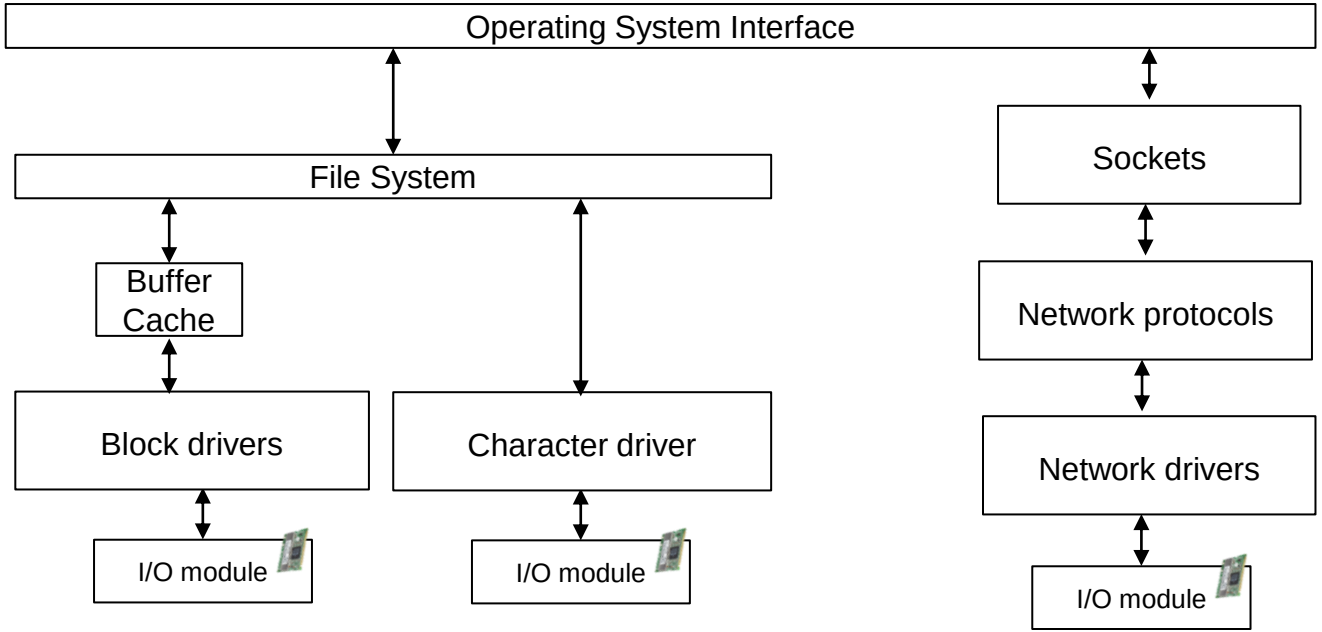
```
alejandro@tesla:~$ mknod /dev/sensor1 c 12 1

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“
```

Simplified logic vision

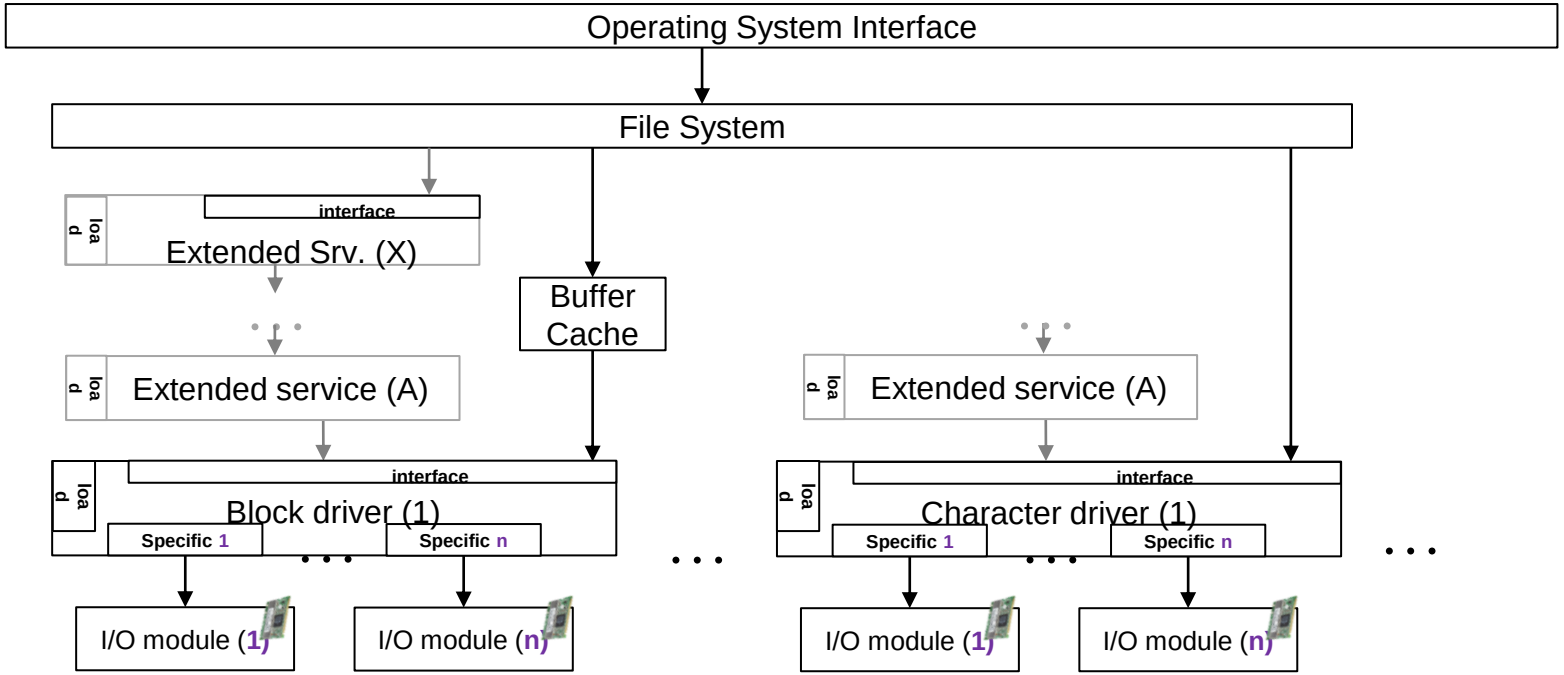
Linux



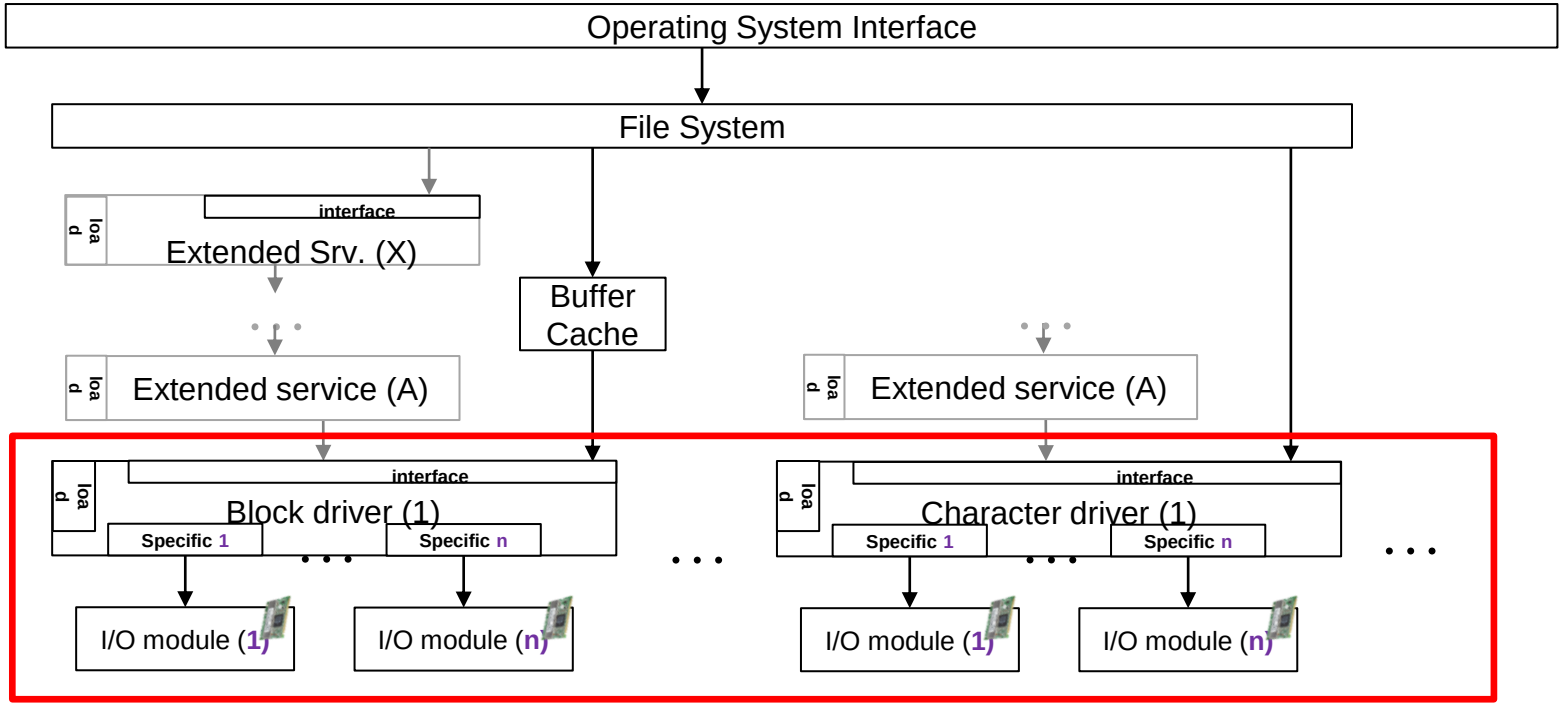
I/O system architecture



Generic structure of the I/O system



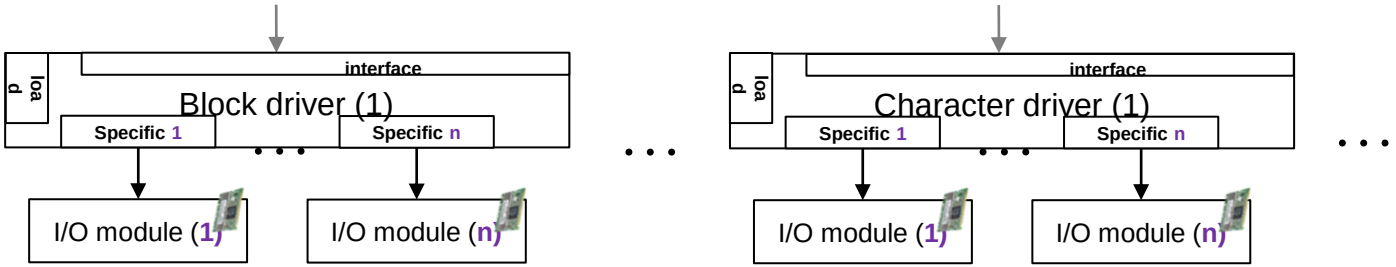
Generic structure of the I/O system



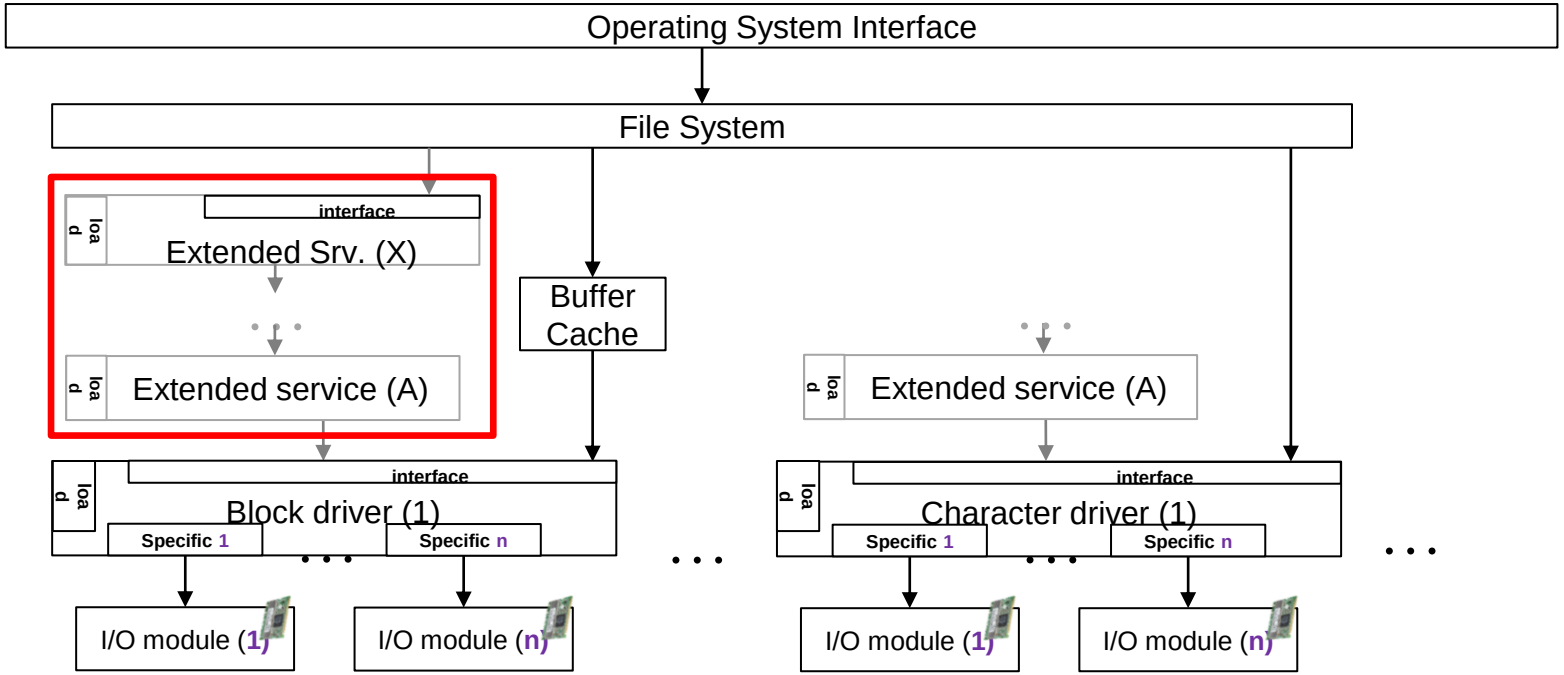
Generic structure of the I/O system

classification of drivers

- The drivers are classified according to the group of devices to which it deals.
  - If two drivers treat the same type of device then the interface is similar.
  - Part of the implementation of the driver is common (code is saved).
- Classically there are three types:
  - Character device: keyboard, modem, etc.
  - Block device: hard disk, tapes, etc.
  - Network device: network card, etc.

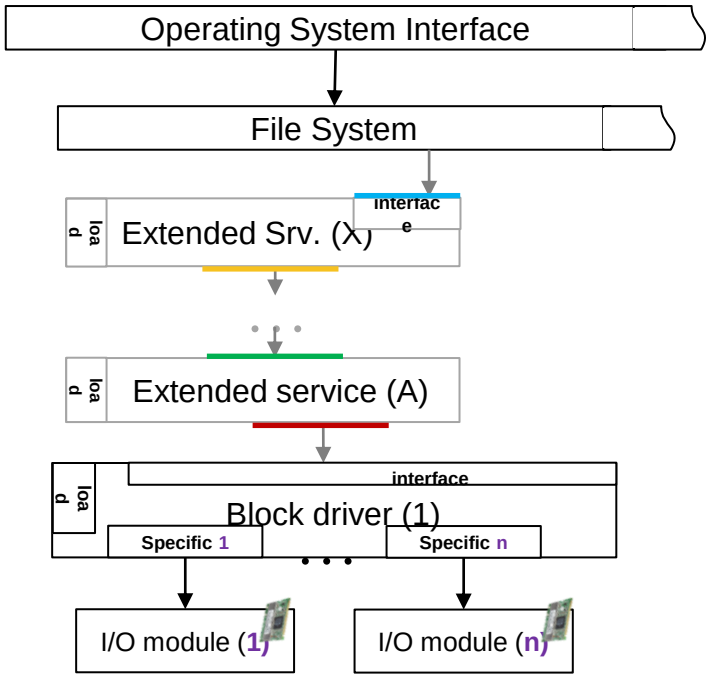


Generic structure of the I/O system



Generic structure of the I/O system

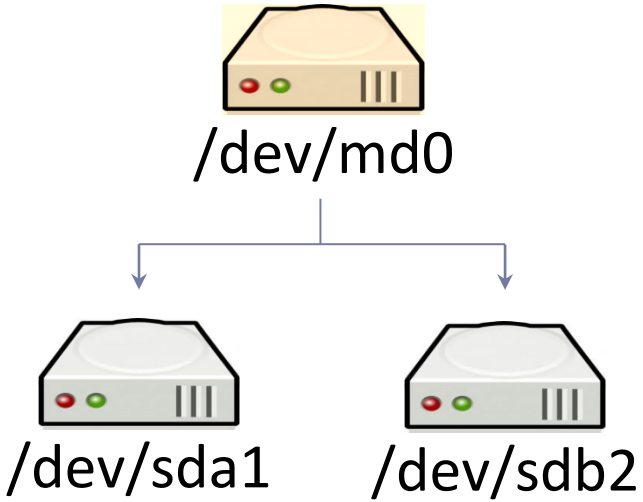
extended services



- **Extended service:**
  - Module that extends a driver to add some kind of functionality.
  - They are stackable with each other.
- **It has, at least, two interfaces:**
  - The service interface is:
    - System call interface.
    - Upper Ext. service interface.
  - The resource interface:
    - I/O module interface
    - Lower Ext. service interface

Extended services

Linux

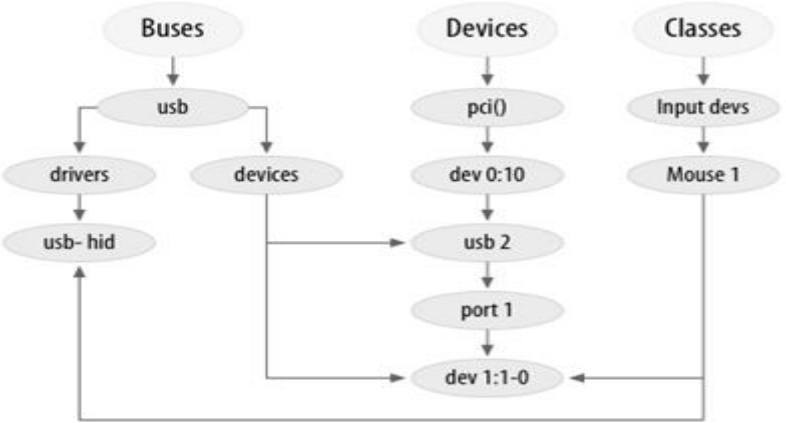


- Example of extended service:
  - md (*multiple disks*)
- Combine multiple hard drives, or partitions (or volumes) into a single virtual disk.
  - `mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sda1 /dev/sdb1`

Drivers hierarchy

Linux

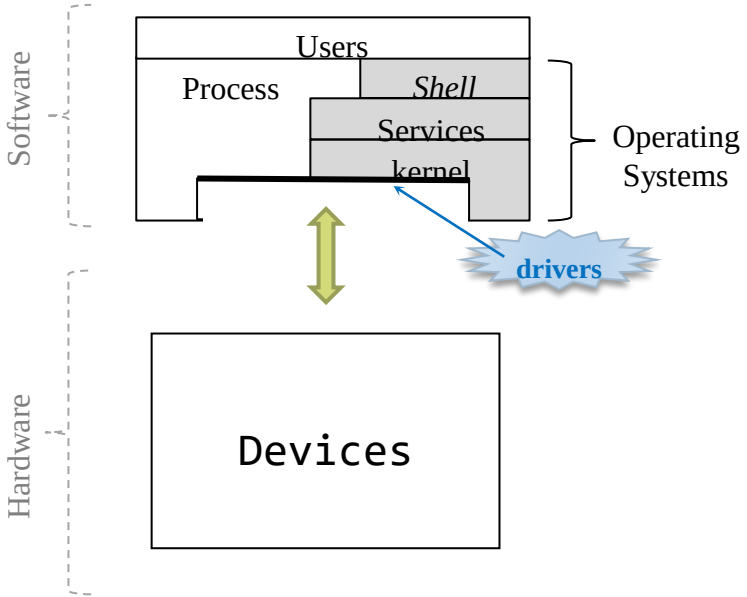
- The hierarchy of the Linux model is shown in the figure:
  - Buses in the lower level
  - Peripherals in the intermediate level
  - Classes at the highest level



- Access through sysfs:
  - /sys/block: block devices (any bus)
  - /sys/bus: system buses (devices are here)
  - /sys/devices: devices organized by buses
  - /sys/class: kind of devices (audio, network, etc.)
  - /sys/module: registered drivers in the kernel
  - /sys/power: power management
  - /sys/firmware: firmware management (in some types of devices)

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

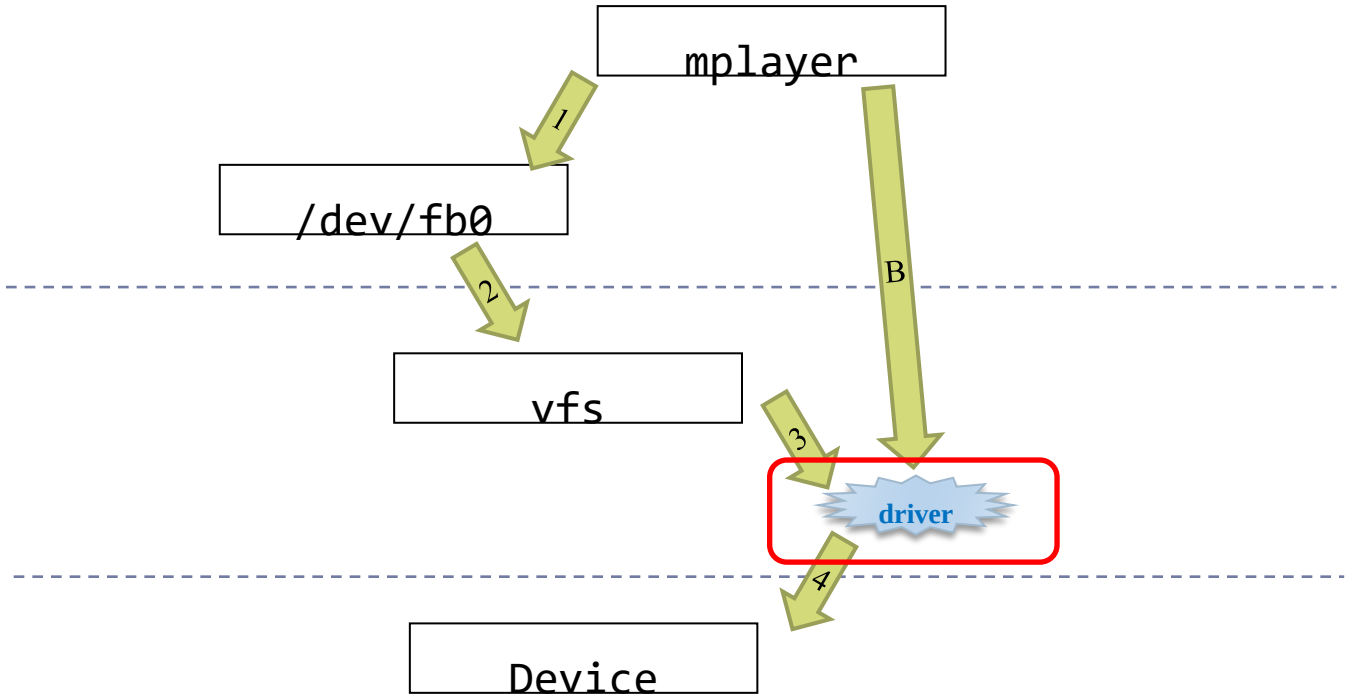
Overview



- Introduction
- Driver framework
- **Structure of one driver**
- Driver design examples

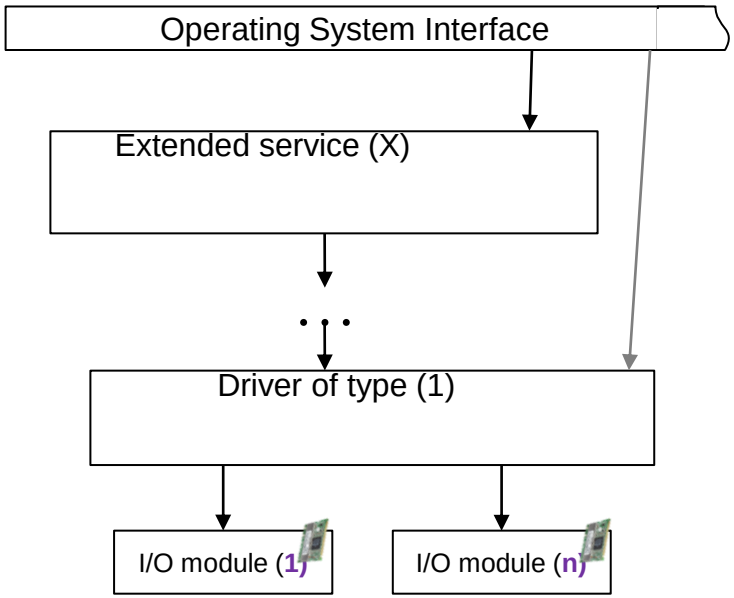
Simplified logic vision

Linux



Basic organization of a driver/ext.srv.

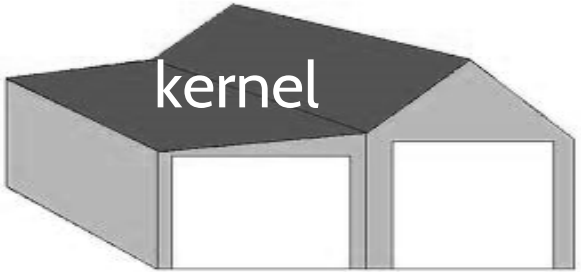
drivers based on kernel modules

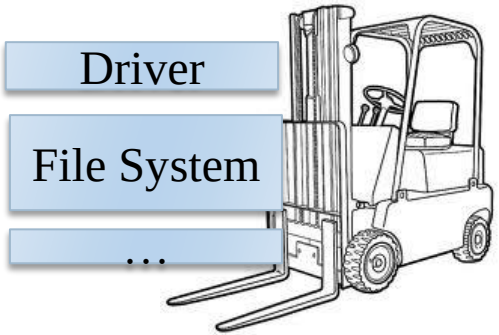


- Not all drivers/ext.serv. are necessary at all times:
  - There are devices that connect / disconnect without turning off the computer (hot-plug devices)
- There are two (combinable) methods for the selection of drivers/ext.serv. to be used:
  - Choose them when the kernel is compiled.
    - When the O.S. boots, the selected drivers/ext.serv. are created.
  - Choose them while kernel is running (dynamic linking).
    - They could be created at some point in the execution of the operating system.

Modules to extend the kernel

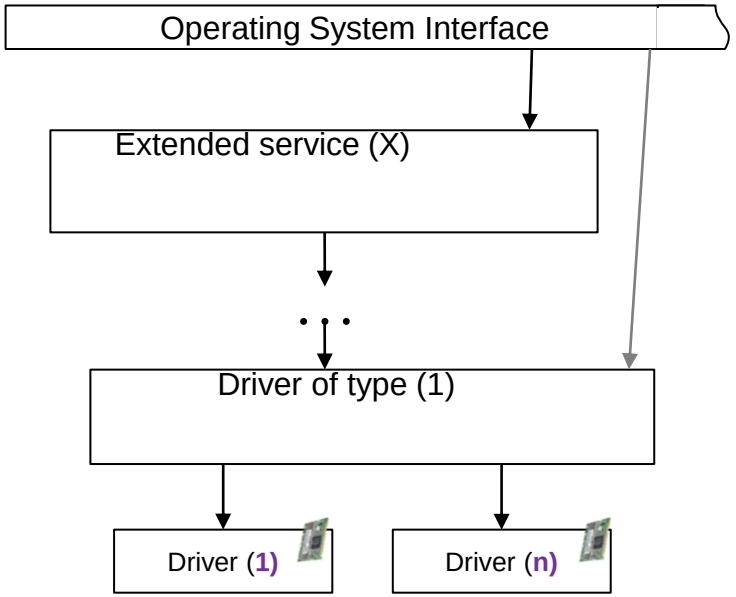
- The modules are used not only for the drivers of the devices, currently they are also used to add other types of functionality:
  - File systems, network protocols, extra system calls, etc.





Basic organization of a driver/ext.srv.

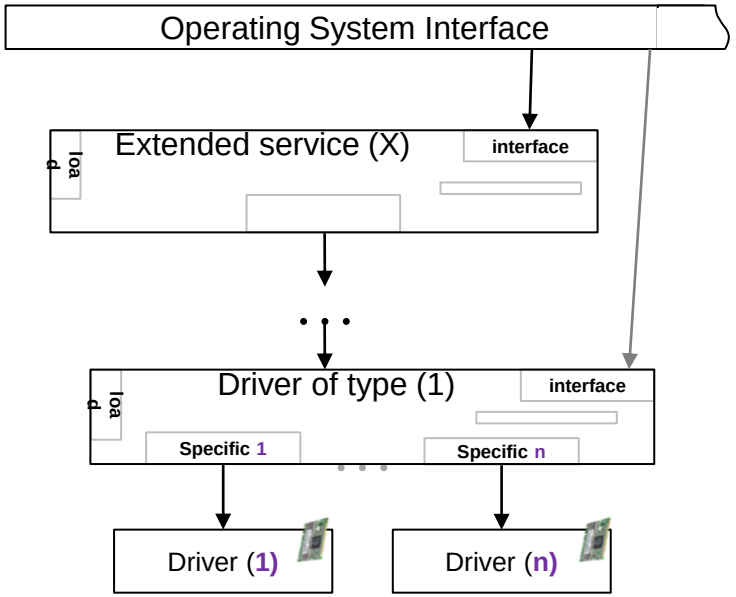
inner parts of a driver module




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

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


/*  Tipo con atributos específicos
    of the device de teclado          */
typedef struct  {
     TipoBufferCaracteres bufferCaracteres;
     TipoListaBCP listaProcessBloqueados;
} TipoDataPropiosDispositivo_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 desload de módulos */
int loadModulo_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) ;
```

Basic organization of a driver/ext.srv.

inner parts of a driver module




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

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


/*  Tipo con atributos específicos
    of the device de teclado          */
typedef struct  {
     TipoBufferCaracteres bufferCaracteres;
     TipoListaBCP listaProcessBloqueados;
} TipoDataPropiosDispositivo_teclado;
```

interface

```
/*  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) ;
```

specific X

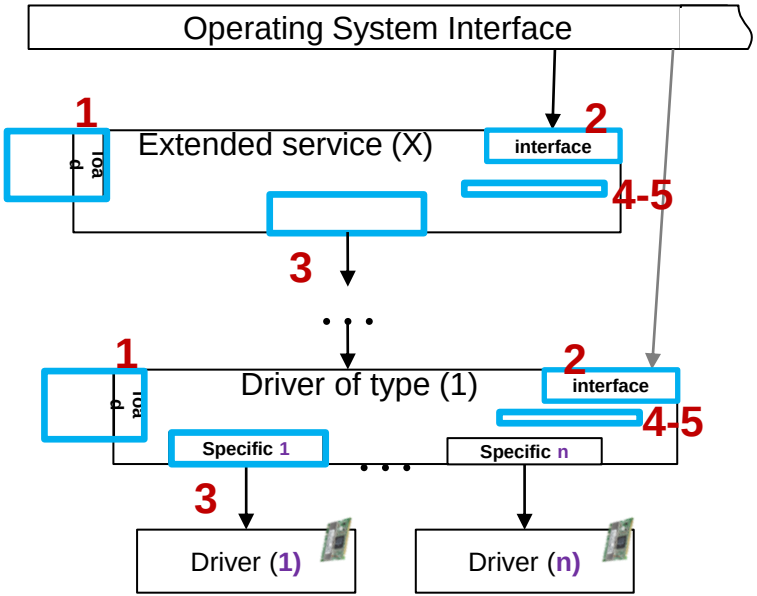
```
/*  Dispositivo de teclado  */
int    interrupcionHW_teclado   (int descDispositivo) ;
void  interrupcionSW_teclado   (int descDispositivo) ;
int peticionCaracter_teclado    (int descDispositivo,
                                 char *caracter, int operacion) ;
```

load/unload

```
/* Cargar y desload de módulos */
int loadModulo_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) ;
```

Basic organization of a driver/ext.srv.

inner parts of a driver module

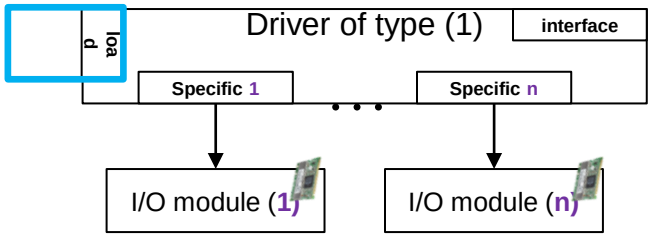


1. Check-in of the driver
2. Interface for system calls
3. Request to the device driver

1. I/O scheduling in Driver
2. Initialization and termination of the driver

Basic organization

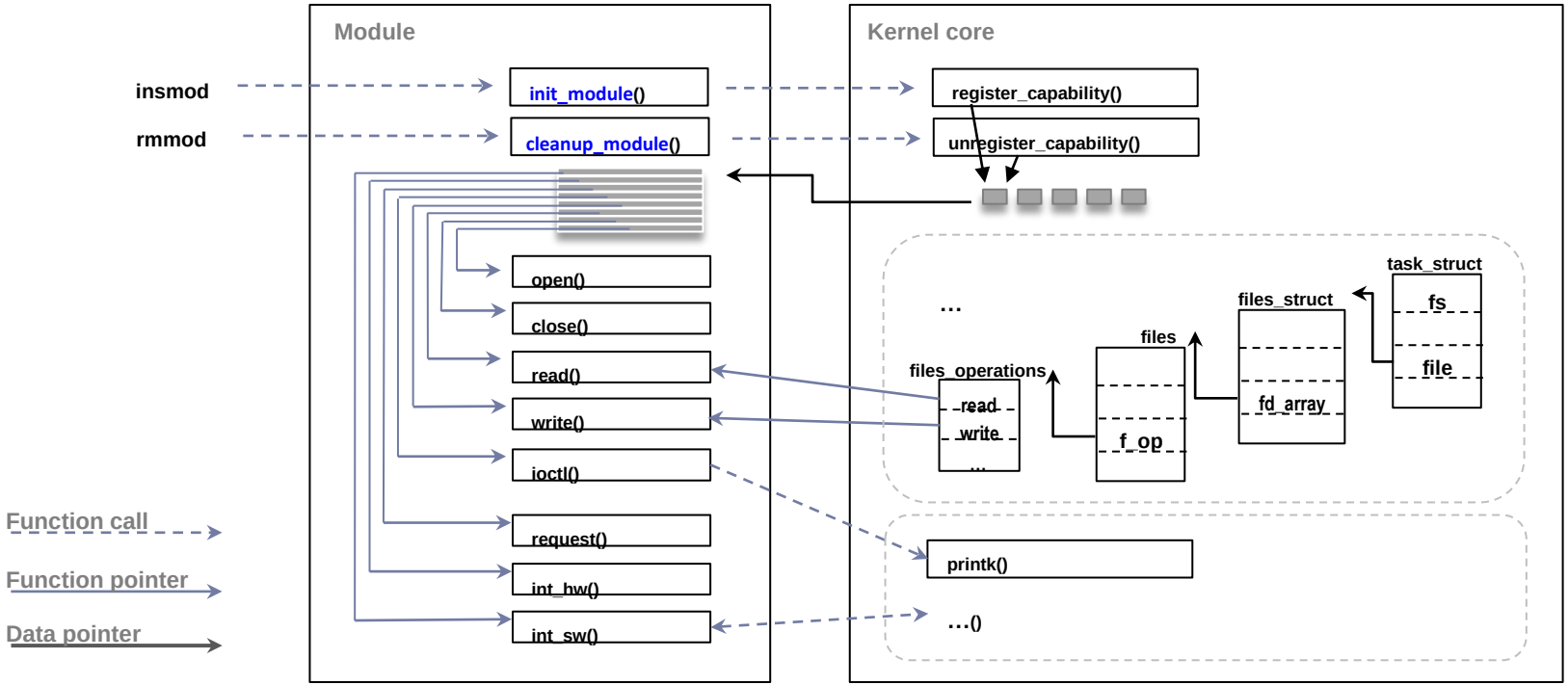
1. driver registration (check-in)



- Table with the loaded drivers:
  - Functions to register drivers.
    - load the associated module for this driver.
  - Functions to deregister drivers.
- Table with detected peripherals:
  - Functions to register the peripheral in the driver, and to register their particular structures/functions.
    - From the driver you have access to the list of peripherals that it manages.
  - Functions to search and deregister (unsubscribe) a peripheral.

Main management structures

Linux



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

check-in process

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 loaded…\n");
    return 0;
}

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

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

check-in process
Linux

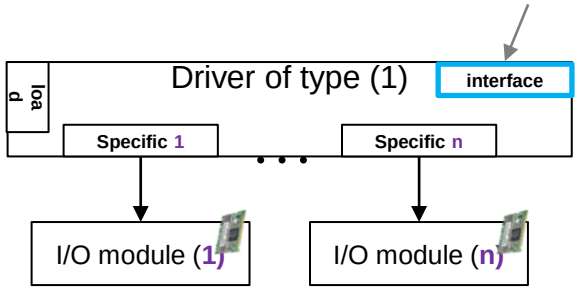
dso/Makefile

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

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

Basic organization

2. interface for system calls



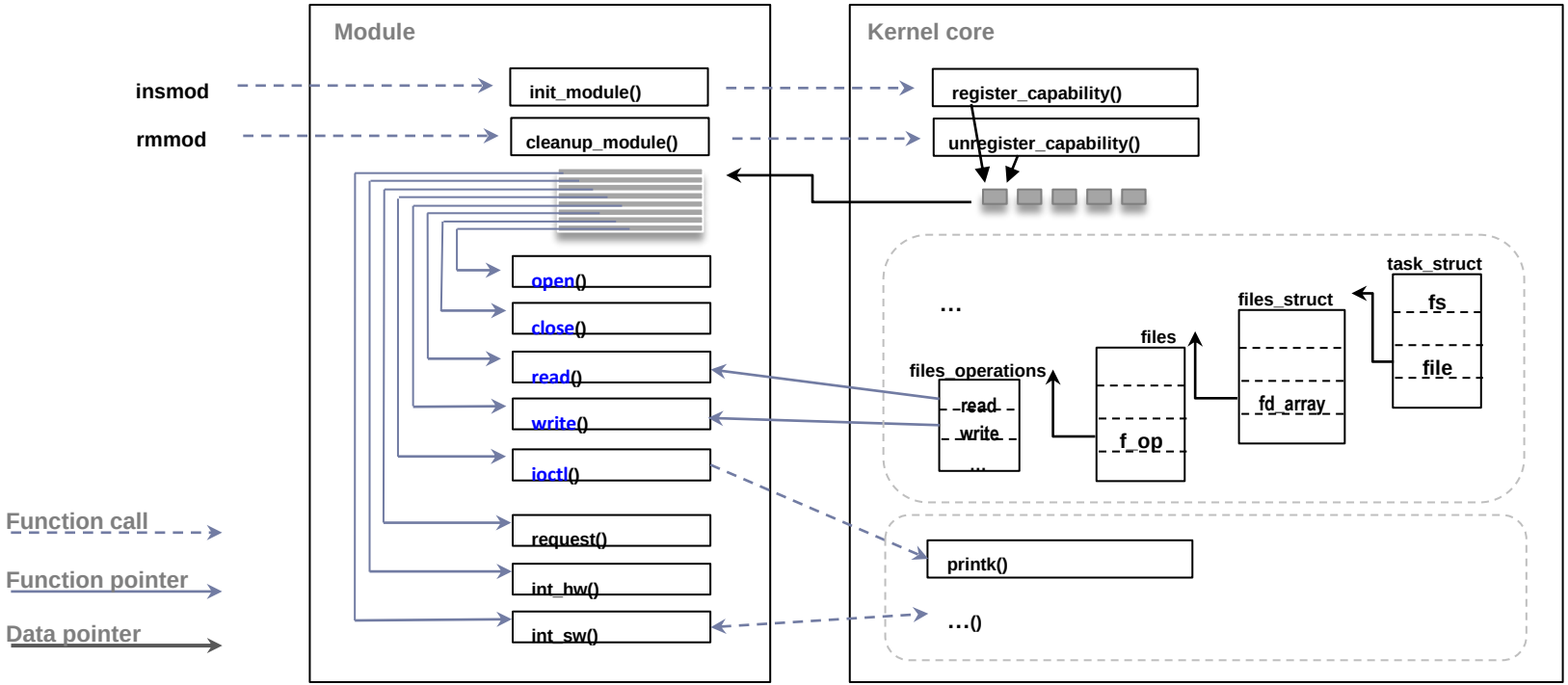
- Interface for system calls:
  - Set of functions that a driver provides to access the peripheral.
- Features:
  - Standardization:
    - If a hardware device is valid for a task, the user or service program of the O.S. that performs it must be able to use it without modifying its code.
  - Use of common and reduced interfaces of system calls:
    - Creating a new call is more expensive than reusing existing calls.

Interface for system calls
Linux

- System calls to set device work session:
  - Open (name, flags, mode)
  - Close (descriptor)
- System calls to interchange data with device:
  - Read (descriptor, buffer, size)
  - Write (descriptor, buffer, size)
  - Lseek (descriptor, offset, whence)
- Generic system calls for devices:
  - Ioctl (descriptor, id_operation, pointer_parameters)
    - Allows the execution of any service with any parameters.
    - The operations must be made public in some way so that there are no conflicts between different drivers.

Main management structures

Linux



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

Interface for system calls

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 <linux/uaccess.h>  /* copy_from/to_user */


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

Interface for system calls

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, /* A.K.A. close */
        read:    test2_read,
        write:   test2_write
    };

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

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

Interface for system calls

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");
}
```

Interface for system calls

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 0; /* SUCCESS */
}

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

Interface for system calls

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;
}
```

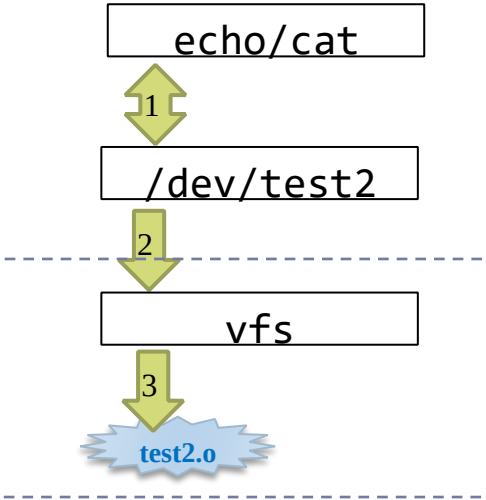
Interface for system calls

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
```



Interface for system calls
Linux

- System calls to set device work session:
  - Open (name, flags, mode)
  - Close (descriptor)
- System calls to interchange data with device:
  - Read (descriptor, buffer, size)
  - Write (descriptor, buffer, size)
  - Lseek (descriptor, offset, whence)

+

Iread
Iwrite
Wait
Ready

- Generic system calls for devices:
  - Ioctl (descriptor, id_operation, pointer_parameters)
    - Allows the execution of any service with any parameters.
    - The operations must be made public in some way so that there are no conflicts between different drivers.

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

Interface for system calls

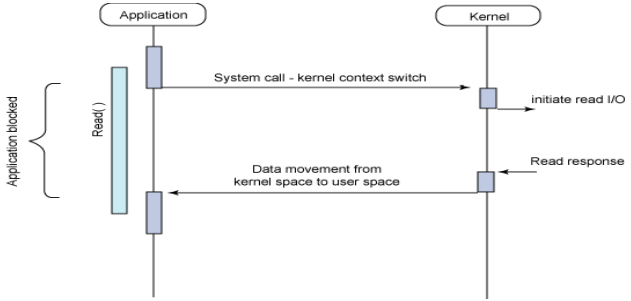
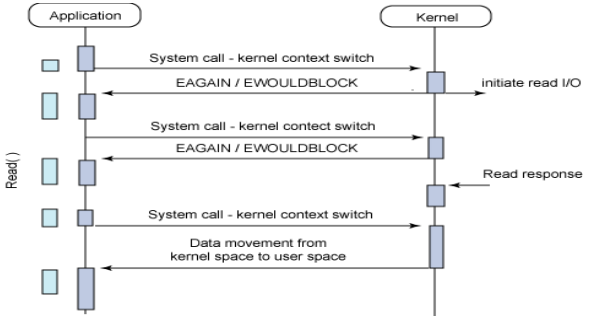
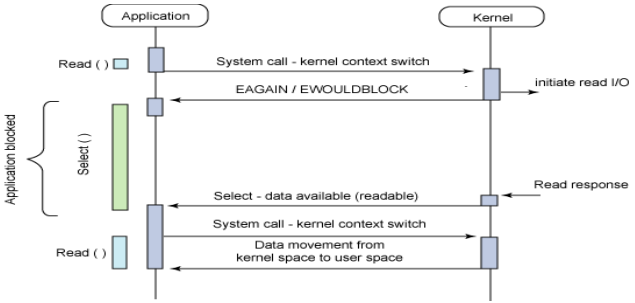
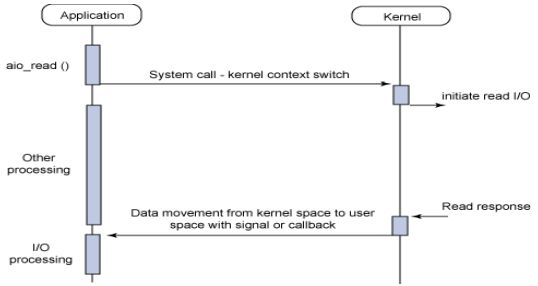
Summary of basic I/O modes in Linux

	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/

Interface for system calls

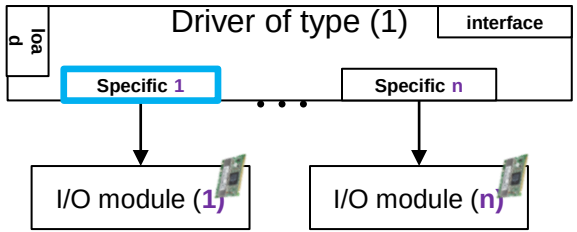
Summary of basic I/O modes in Linux

	Blocking (notification)	NON-Blocking (notification)
Synchronous (request)		
Asynchronous (request)		

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

Basic organization

3. request to the device driver

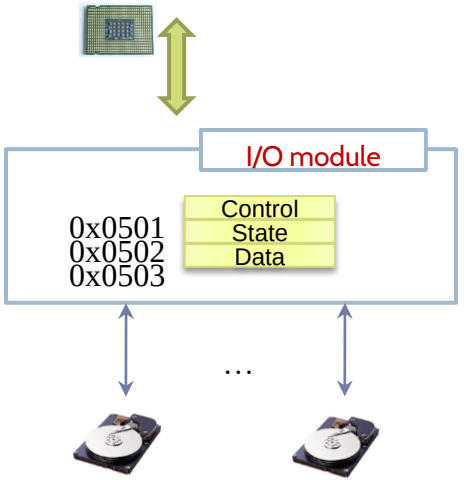


- Requires implementing up to two functions:
  - Function to request the order:
    - Requested by a system call.
  - Function to handle the device interrupt (at the end of the order):
    - It is executed upon receiving the interrupt.
- Necessary to adapt to the hardware type of the device driver:
  - Fast devices or real-time devices.
  - Devices with dependent/independent requests
  - Proactive or reactive devices

Necessary adaptation…

Summary of the fundamental characteristics

- Transfer unit
  - Block
  - Character
- Addressing
  - Memory-mapped
  - Port-mapped
- CPU-module interaction
  - Direct I/O
  - Interrupt I/O
  - DMA I/O
- Main types of protocols
  - Individual request-response
  - Shared request-response
  - Only request
  - Only interrupt



CPU-I/O module interaction

Direct I/O, interrupt I/O, and DMA I/O

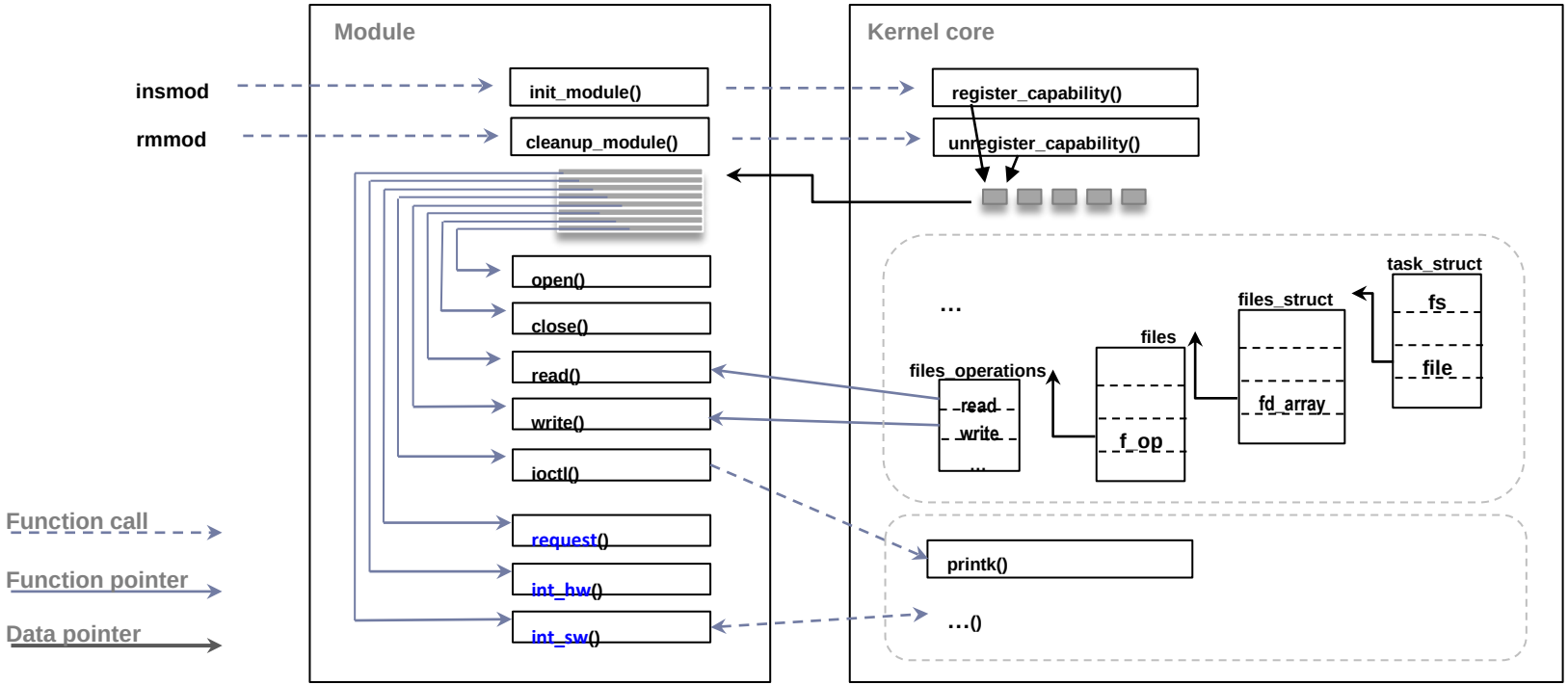
```
request:
for (i=0; i<100;i++)
{
    // read from element 10
    out(0x504, 10) ;
    out(0x500, 0) ;

    // waiting loop
    do {
      in(0x508, &status) ;
    } while (0 == status) ;

    // read data
    in(0x50C, &(Data[i])) ;
}
```

```
request:
    out(0x500, 0) ;
    p.neltos = 100;
    p.counter = 0;
    out(0x504, 10) ;
    // V.C.S.

INT_05:
    in(0x508, &status) ;
    in(0x50C, &p.Data[p.counter]) ;
    if (p.counter < p.neltos) {
        out(0x504, 10+p.counter) ;
        out(0x500, 0) ; // leer
        p.counter++ ;
    } else {   // requested process now ready   }
    ret_int # restore registers & return
```

```
request:
    out(0x500, 0) ;
    out(0x500, Data) ;
    out(0x500, 100) ;
    out(0x504, 10) ;
    // V.C.S.

INT_05:
    // read State y Data
    in(0x508,  &status) ;
    in(0x50C,  &status) ;

    // requested process now ready
    ret_int # restore registers & return
```

Main management structures

Linux



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

Communication with the driver

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;
```

Communication with the driver

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;
```

Communication with the driver

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);
```

Communication with the driver

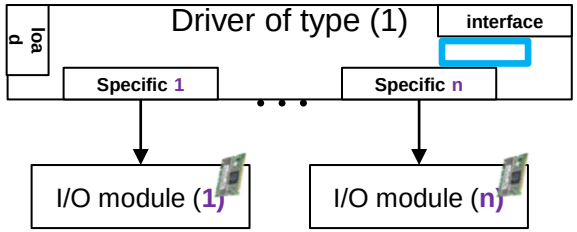
Linux

dso/Makefile

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

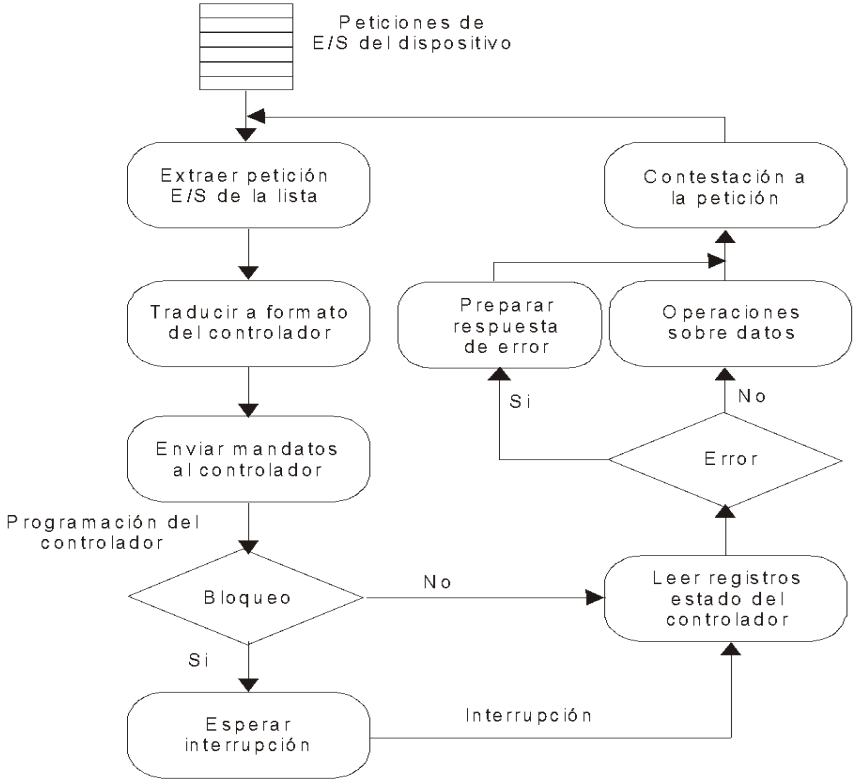
```
make -C /usr/src/linux M=`pwd` modules
tail –f /var/log/syslog &
insmod test3.ko
ls
rmmod test3
```

Basic organization
4. /O planning in the driver

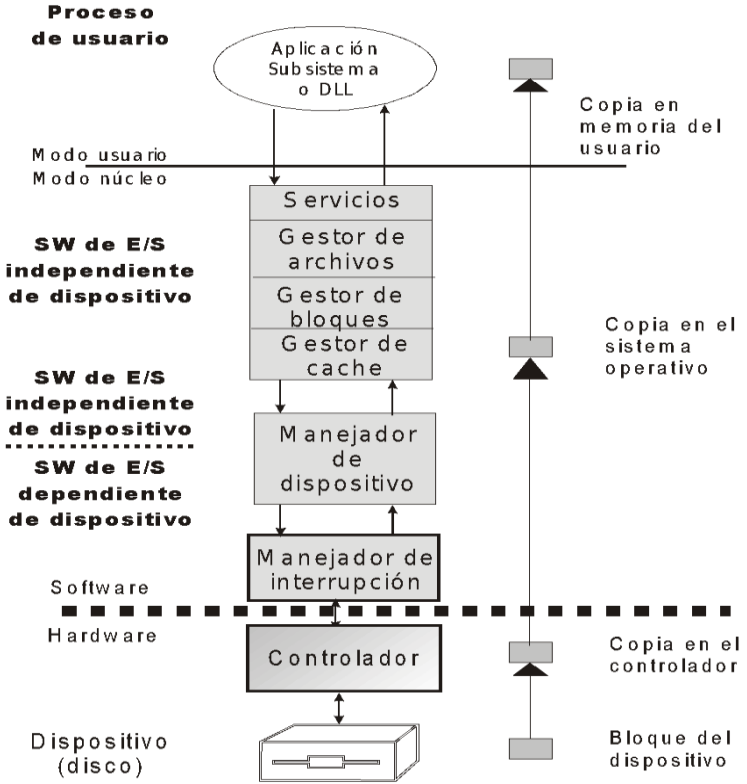


- When there are several requests to a device, these are kept in a queue of requests.
  The driver usually has an I/O scheduler that allows you to plan the requests in a way that minimizes the time of attention to them.
  - The disk blocks are planned to minimize the time spent moving the disk heads.
- The I/O scheduler usually performs at least two basic operations:
  - Sorting: the requests are inserted in a list according to some criterion.
  - Fusion: two consecutive small requests are transformed into a single request.

I/O scheduling in Driver

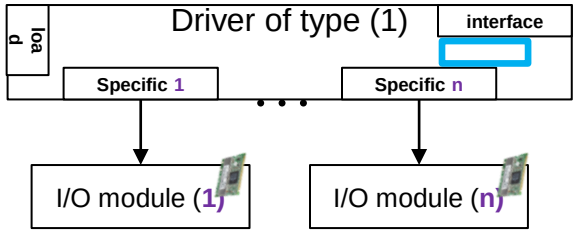


workflow of an I/O operation



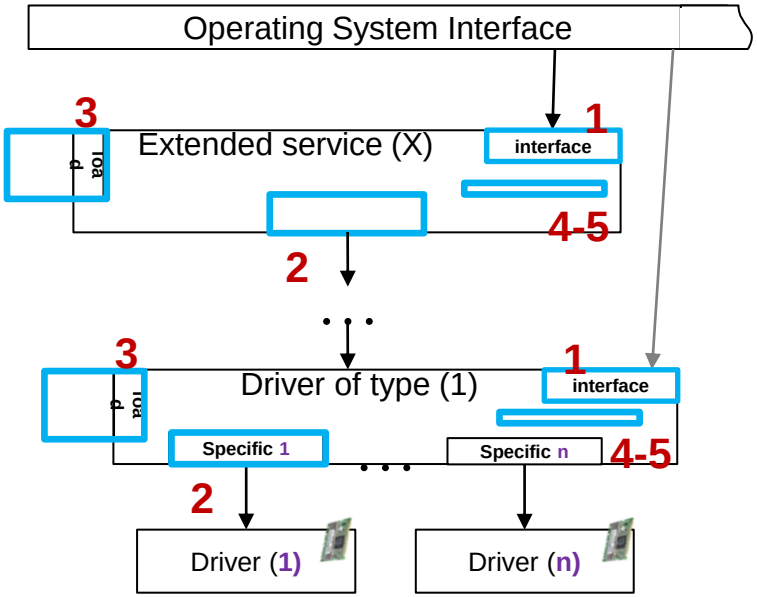
Basic organization

5. initialization and driver completion



- When a driver is being used, it needs a series of associated resources (IRQ, memory buffer, etc.)
- To control the allocation of resources you can follow the following scheme:
  - A counter maintains the number of processes that will work with a device.
  - Each time a new process operates with a device, the counter is increased, and when it ceases to operate it is decremented.
  - When the counter goes to 1, the resources are assigned to the driver.
  - When the counter is set to 0, all resources are released.

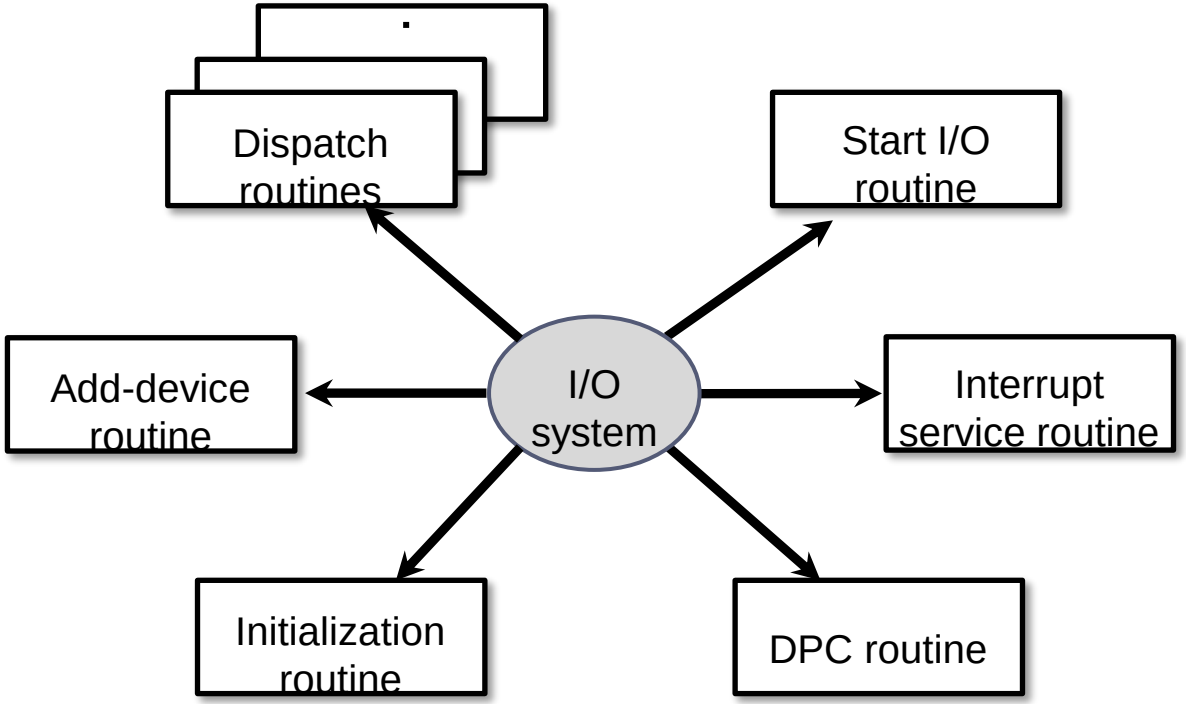
Basic organization
summary



1. Interface for system calls
2. Request to the device driver
3. Check-in of the drivers

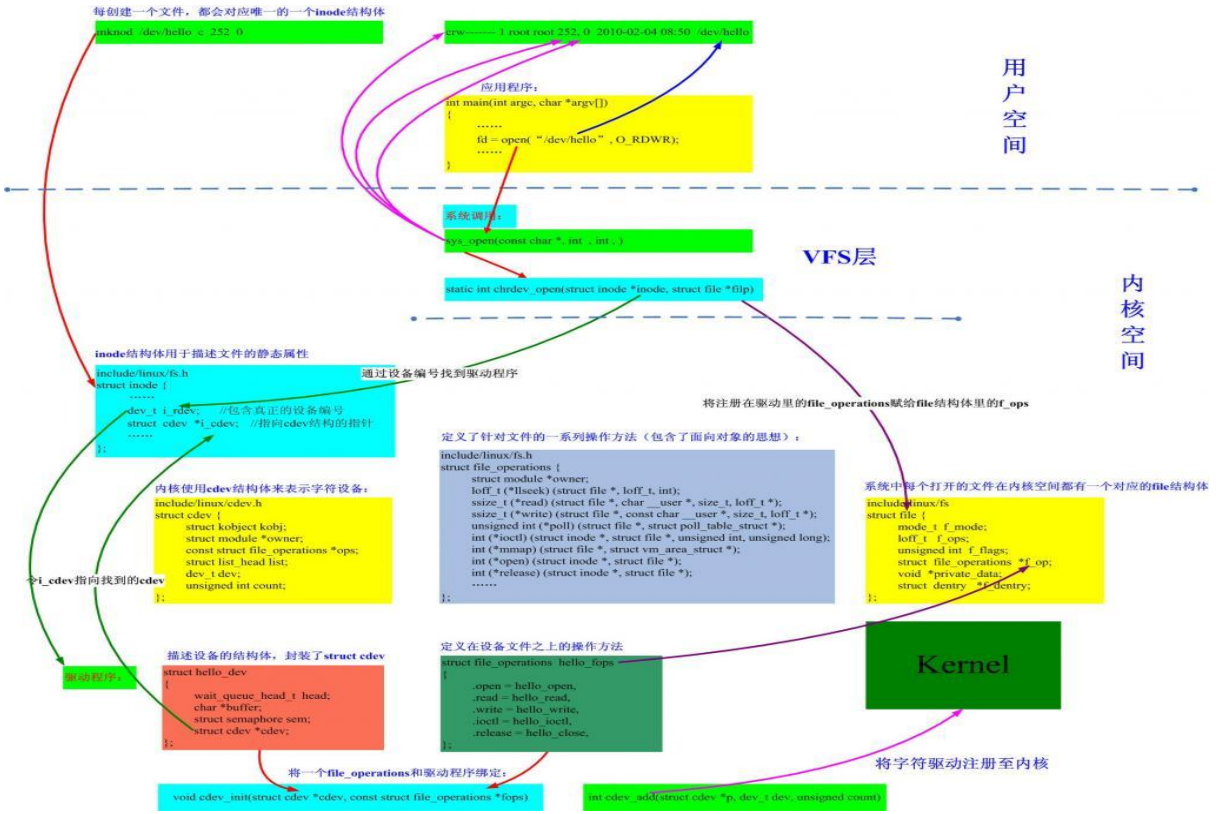
1. I/O scheduling in Driver
2. Initialization and termination of the driver

Windows 2000
driver subroutines



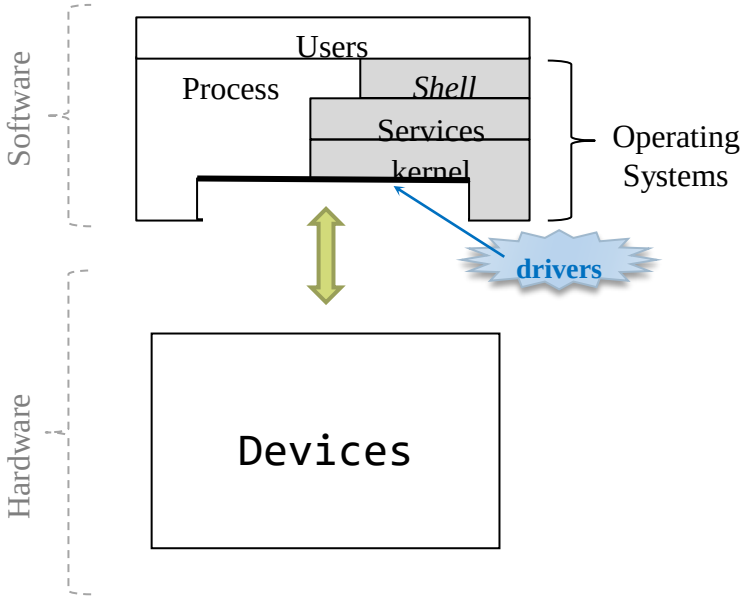
Data structures

Linux



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

Overview



- Introduction
- Driver framework
- Structure of one driver
- **Driver design examples**

Basic organization

Ejemplos con distintos tipos de dispositivos

- Fast device (no v.c.s.)
  - Only request 
  - Only interrupt 
- Slow device (possible v.c.s.)
  - Independent requests   
  - Shared requests  

Request to the device driver

fast (output)

Screen

- Fast device

- Request (data):
  - Copy data into a buffer

- Interrupt handler of the device:

Request to the device driver
fast (input)

Clock

- Fast device

- Request (data):

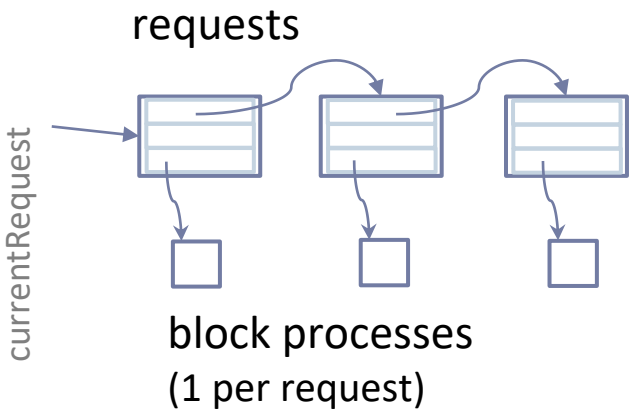
- Interrupt handler of the device:
  - Ticks = Ticks + 1

Request to the device driver

slow / independent (output)

Printer

- Slow device
- Independent requests



- Request (data):
  - Create one request
  - Copy data into request->intermediate_buffer
  - If no printing data
    - Start printing the request
  - Block + to execute another process

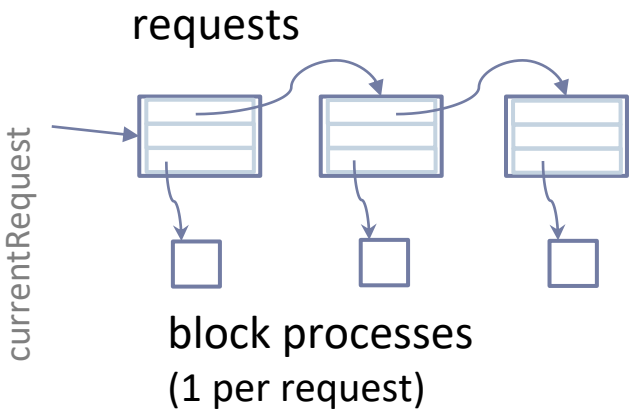
- Interrupt handler of the device:
  - Update process state into "ready"
  - If there is any request enqueue
    - Start printing next request

Request to the device driver

slow / independent (input)

Scanner

- Slow device
- Independent requests



- Request (data):
  - Create one request
  - If scanner is inactive
    - Start scanning request
  - Block + Execute another process
  - Copy from intermediate_buffer into the user buffer

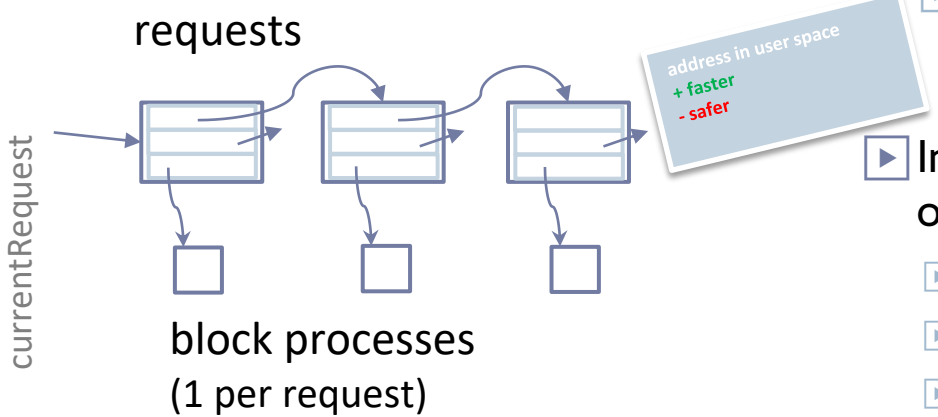
- Interrupt handler of the device:
  - Insert data into the intermediate_buffer
  - Update process state into "ready"
  - If there are requests enqueued
    - Start scanning next request

Request to the device driver

slow / independent (input)

Scanner

- Slow device
- Independent requests



- Request (data):
  - Create one request
  - If scanner is inactive
    - Start scanning request
  - Block + Execute another process
  - ~~Copy from intermediated_buffer into the user buffer~~
- Interrupt handler of the device:
  - Copy data to the user buffer
  - Update process state into "ready"
  - If there are requests enqueued
    - Start scanning next request

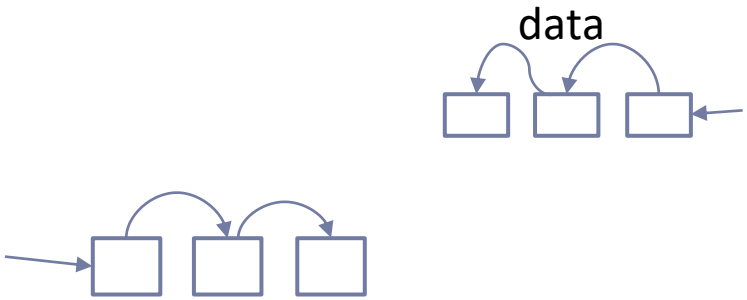
Request to the device driver

slow / independent (input)

Keyboard

- Slow device
- Independent requests

- Request (data):
  - If there is NOT data
    - "Block" + Execute another process
  - Copy data from a buffer



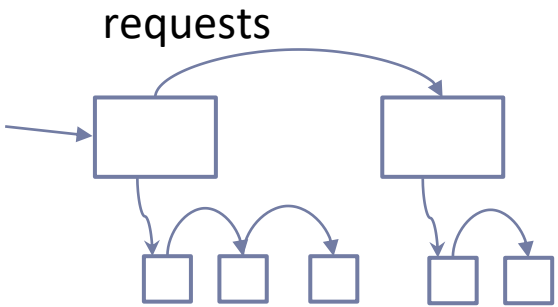
- Interrupt handler of the device:
  - Copy data into a buffer
  - If there is a blocked processes
    - 'wake-up' the first one (set its state to ready)

Request to the device driver

slow / shared (output)

Disk

- Slow device
- Shared requests



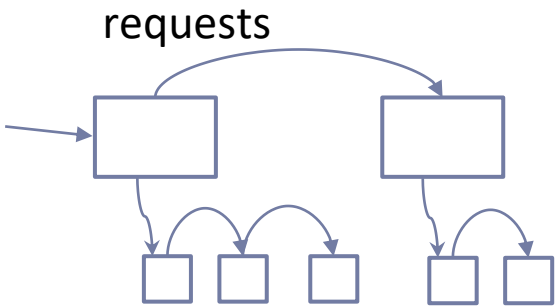
- Request (data):
  - If other process already requested the same
    - Update data
    - Block by this request
  - Otherwise
    - Create a new request
    - Enqueue the request
    - Block by this request
- Interrupt handler of the device:
  - Unblock all processes blocked waiting for the completed request
  - If there are enqueued requests
    - Issue the next request

Request to the device driver

slow / shared (input)

CDROM

- Slow device
- Shared requests



- Request (data):
  - If other process already requested the same
    - Update data
    - Block by this request
  - Otherwise
    - Create a new request
    - Enqueue the request
    - Block by this request
  - Copy the read data
- Interrupt handler of the device:
  - Copy data into a buffer
  - Unblock all processes blocked waiting for the completed request
  - If there are enqueued requests
    - Issue the next request

ARCOS Group
Computer Science and Engineering Department
Universidad Carlos III de Madrid

Lesson 3c

process, devices, drivers, and extended services

Operating System Design
Degree in Computer Science and Engineering, Double Degree CS&E + BA