Taller de Sistemas Operativos

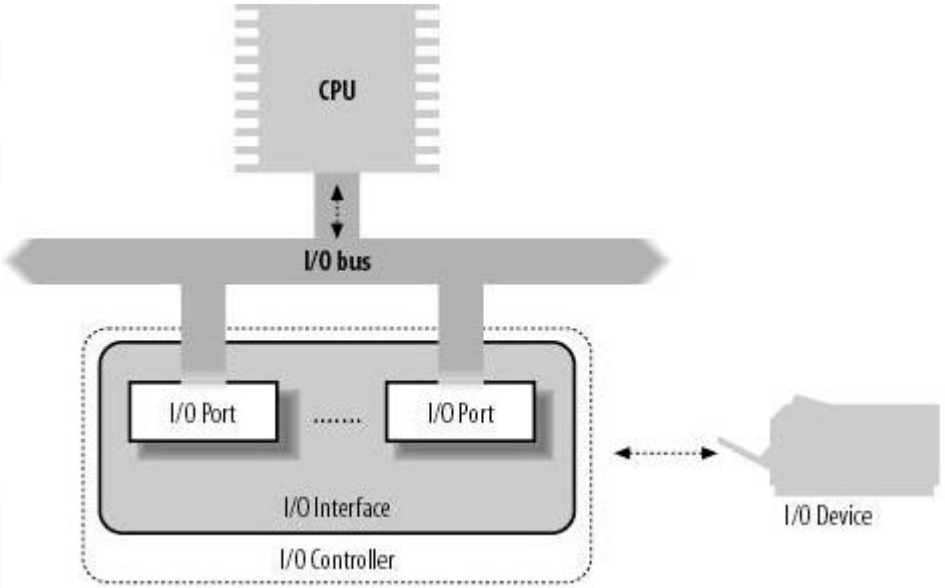
Device Driver Model

Arquitectura de I/O

- Para que una computadora funcione correctamente, deben ser provistos “data paths” para permitir que la información fluya entre la CPU(s), RAM y los dispositivos de I/O.
- Cualquier computadora tiene un system bus que conecta los dispositivos de HW internos.
- Dos buses de alta velocidad se encuentran dedicados a las transferencias de datos desde / hacia memoria.
- Un dispositivo se encuentra conectado a uno y solo un bus.
- El “data path” que conecta la CPU a un dispositivo de I/O se denomina bus de I/O.
- x86 utiliza 16 pines de dirección para direccionar una I/O y 8,16 o 32 de los pines de datos para transferir datos

Arquitectura de I/O

- El bus de I/O se encuentra conectado a cada dispositivo de I/O por una jerarquía de HW que incluye: puertos de I/O, interfaces y controladores de dispositivos.



Puertos de I/O

- Cada dispositivo conectado al bus de I/O tiene su serie de direcciones de I/O llamados puertos de I/O
- Existen cuatro instrucciones de *assembler* `in`, `out`, `ins`, `outs` para leer y escribir de un puerto de I/O
- Los puertos de I/O pueden ser mapeados directamente en memoria y utilizarse instrucciones *assembler* para mover datos en memoria
- Se incluyen el kernel funciones auxiliares para simplificar la I/O

  - `Inb(), Inw(), Inl()`
  - `Inb_p(), Inw_p(),Inl_p()`
  - `Outb(), Outw(), Outl()`
  - `Outb_p(), Outw_p(), Outl_p()`
  - `Insb(), Insw(), Insl()`
  - `Outsb(), Outsw(), Outsl()`

Puertos de I/O

- Un recurso representa una entidad asignada a un *device* driver.
- En este caso representa un rango de I/O *ports*
- Se representan como un árbol cuya raíz se encuentra en el nodo `ioport_resource`

Type	Field	Description
`const char *`	`name`	Description of owner of the resource
`unsigned long`	`start`	Start of the resource range
`unsigned long`	`end`	End of the resource range
`unsigned long`	`flags`	Various flags
`struct resource *`	`parent`	Pointer to parent in the resource tree
`struct resource *`	`sibling`	Pointer to a sibling in the resource tree
`struct resource *`	`child`	Pointer to first child in the resource tree

Interfaces de I/O

- Es un circuito entre un grupo de I/O *ports* y el correspondiente *device controller*
- Traduce valores en los I/O *ports* en comandos y datos para el dispositivo
- Este circuito puede ser conectado mediante una línea de IRQ a un PIC para que pueda generar interrupciones en nombre del dispositivo
- I/O interfaces “a medida”. Son destinadas a un HW en particular
  - Interface de teclado
  - Interface de mouse
  - Interface de Red
- I/O interfaces de propósito general. Se usan para conectar distintos dispositivos
  - Puerto paralelo
  - SCSI
  - USB

Device controllers

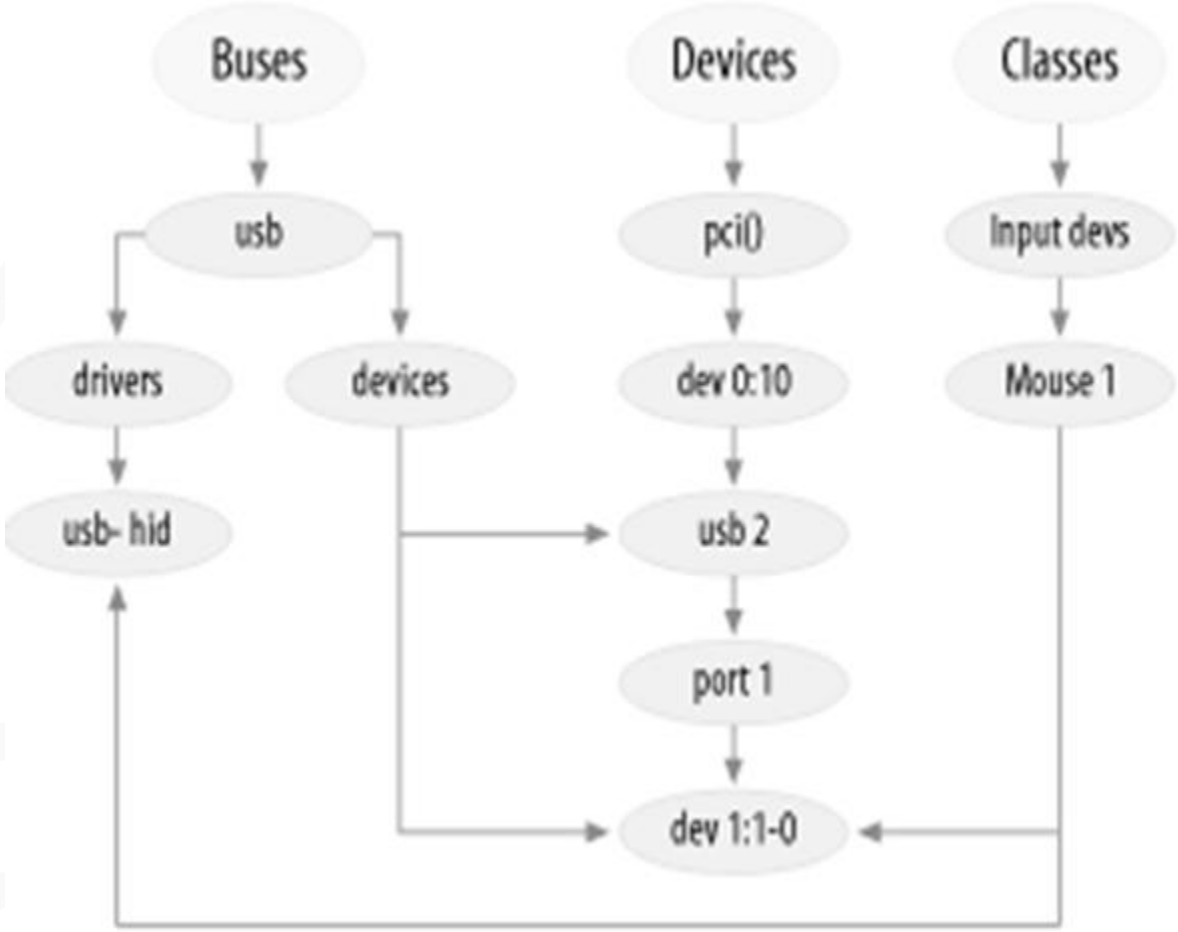
- Un dispositivo complejo requiere un controlador para manejarlo
  - Interpreta comandos de alto nivel desde la *I/O interface* y hace que el dispositivo ejecute acciones específicas
  - Interpreta y convierte las señales eléctricas recibidas del dispositivo y modifica el valor del registro de estado.

- Un *device controller* típico es el controlador de disco que recibe comandos como “grabar este bloque” y lo traduce en operaciones de disco como “posicionar la cabeza en una pista”, “grabar los datos en la pista”, etc

Device Driver Model

- Uno de los objetivos de 2.5 fue la creación de un modelo unificado de dispositivos en el kernel
- En kernel previos no había una estructura de datos donde se podía ver como estaba compuesto el sistema
- El modelo de dispositivos de 2.6 provee una abstracción de cómo está compuesto el sistema y nos permite realizar algunas tareas
  - Power management y system shutdown
  - Comunicación con user space (mediante el filesystem sysfs)
  - Dispositivos HotPlug
  - Clases de dispositivos (device classes)
  - Ciclo de vida de los objetos

Device Driver Model



El filesystem sysfs

- Es un *filesystem* especial similar a `/proc` montado en /`sys`
- El objetivo es exponer la relación jerárquica entre los componentes del *ddm* (*device driver model*)
- Los directorios de primer nivel son:
  - `block`. Block devices independientemente del bus al que se encuentran conectados
  - `devices`. Todos los dispositivos reconocidos, organizados según el bus al que se encuentran conectados.
  - `drivers`. Los *device drivers* registrados en el kernel
  - `class`. Los tipos de dispositivos en el sistema (placas de video, red, sonido, etc)
  - `power`. Archivos para manejar el estado de energía de algunos dispositivos
  - `firmware`. Archivos para manejar el firmware de algunos dispositivos
- La relación entre componentes de dispositivos son expresadas en términos de *symlinks*
- El objetivo de *sysfs* es representar atributos de drivers y dispositivos

El filesystem sysfs

- Por ejemplo el 1er disco IDE puede figurar en `sysfs` como:

  /sys/devices/pci0/00:11.1/ide0/0.0

- También puede aparecer (como link simbólico) bajo `subsystems`:

  /sys/block/hda/device

  /sys/bus/ide/devices/0.0

- Adicionalmente el controlador IDE puede ser encontrado como:

  /sys/bus/pci/devices/0.11.1

  /sys/bus/pci/drivers/VIA IDE/00:11.1

Kobjects

- Es la estructura principal del ddm.
- Cada *kobject* corresponde a un directorio en `sysfs` (/sys)
- Se encuentran “incrustados” dentro de objetos más grandes llamados “*containers*”
- Incrustar un *kobject* dentro de una estructura nos ayuda a
  - Mantener un contador de referencias para el *container*
  - Mantener una lista jerárquica o conjunto de *containers*
  - Proveer una vista desde “*user mode*” de los atributos del *container*

Type	Field	Description
`char *`	`k_name`	Pointer to a string holding the name of the container
`char []`	`name`	String holding the name of the container, if it fits in 20 bytes
`struct k_ref`	`kref`	The reference counter for the container
`struct list_head`	`entry`	Pointers for the list in which the kobject is inserted
`struct kobject *`	`parent`	Pointer to the parent kobject, if any
`struct kset *`	`kset`	Pointer to the containing kset
`struct kobj_type *`	`ktype`	Pointer to the kobject type descriptor
`struct dentry *`	`dentry`	Pointer to the dentry of the *sysfs* file associated with the kobject

Kobjects

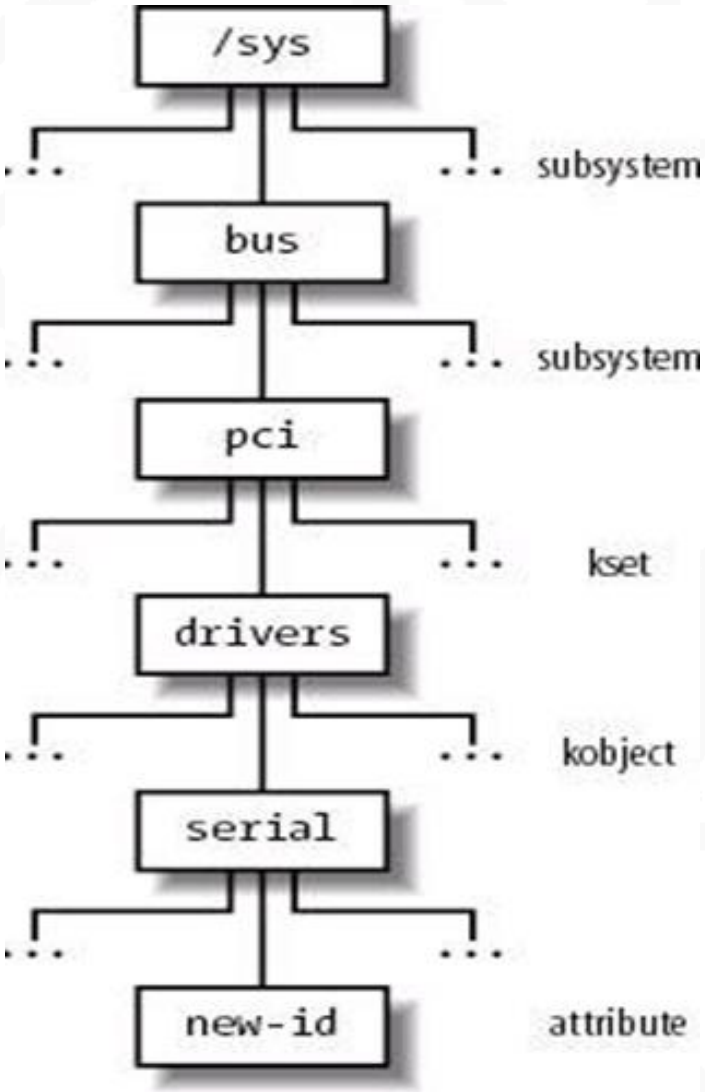
- La estructura `kobj_type` contiene tres campos
  - Un método `release`
  - Un puntero `sysfs_ops` a la lista de operaciones `sysfs`
  - Una lista de atributos por defecto al filesystem `sysfs`
- El campo `kref` consiste en campo de tipo refcount. Se maneja mediante operaciones `kobject_get`() y *`kobject_put`*()
- Los *kobjects* pueden ser organizados en árboles jerárquicos por medio de *ksets*. Un *kset* es una colección de *kobjects* del mismo tipo incluido en el mismo tipo de *container*.

Type	Field	Description
`struct subsystem *`	`subsys`	Pointer to the subsystem descriptor
`struct kobj_type *`	`ktype`	Pointer to the kobject type descriptor of the kset
`struct list_head`	`list`	Head of the list of kobjects included in the kset
`struct kobject`	`kobj`	Embedded kobject (see text)
`struct kset_hotplug_ops *`	`hotplug_ops`	Pointer to a table of callback functions for kobject filtering and hot-plugging

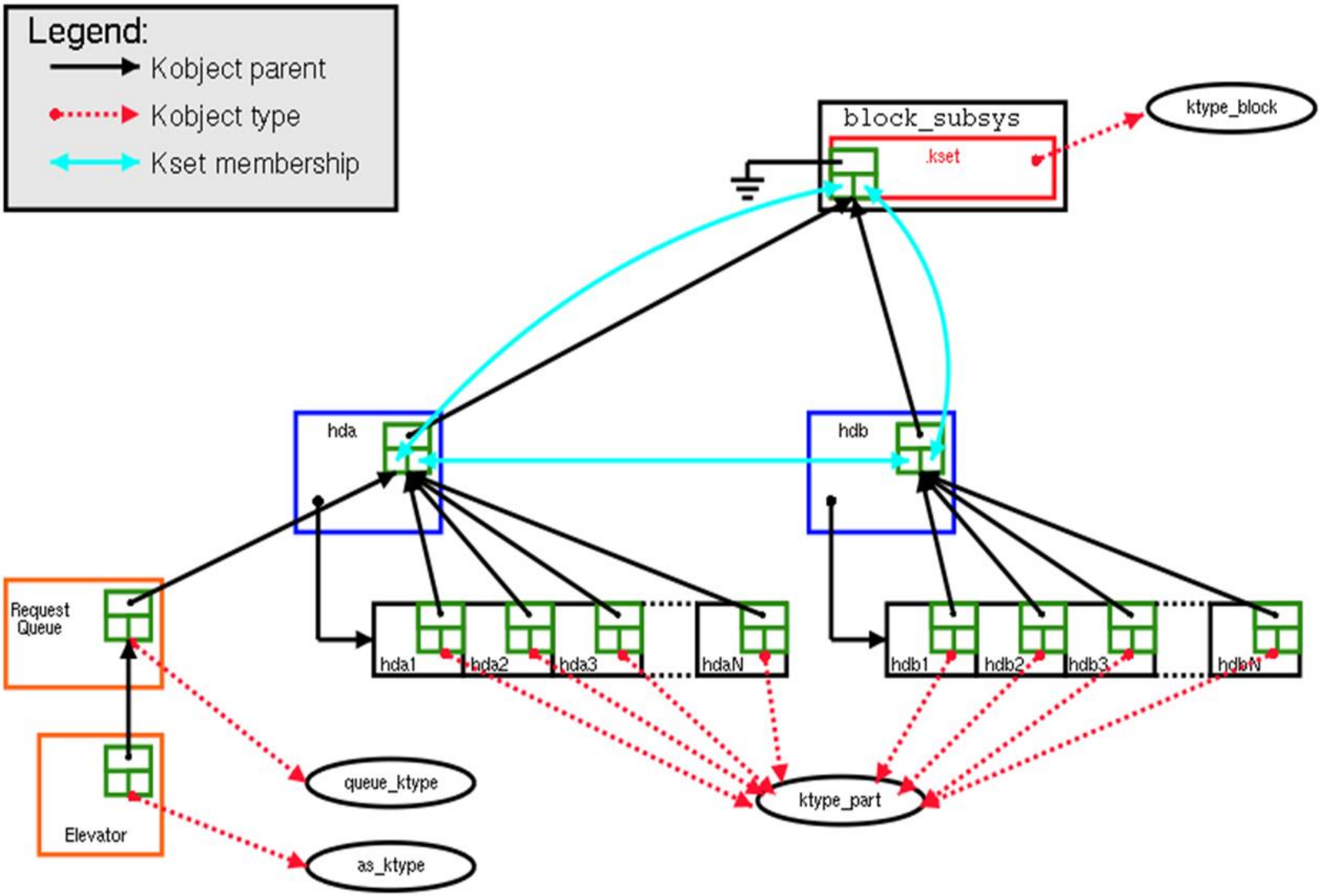
Kobjects

- Existen colecciones de *ksets* llamados subsistemas
- Un subsistema incluye:
  - `kset`. Un *kset* embebido que guarda los *ksets* incluidos en el subsistema
  - `rwsem`. Un semáforo *read-write* que protege todos los *ksets* y *kobjects* incluidos en el subsistema
- Para que aparezca un *kobject*, *kset* o subsistema en `sysfs`
  - Se debe registrar
  - Los directorios asociados con un *kobject* aparecen en el directorio del *kobject* padre
  - Los directorios de *kobjects* en el mismo *kset*, aparecen en el directorio del *kset*
- La estructura de `sysfs` representa la jerarquía entre los *kobjects* registrados
- Otras relaciones entre objetos representados en `sysfs` se realizan mediante links simbólicos

Kobjects


/sys
subsystem
bus
subsystem
pci
kset
drivers
kobject
serial
new-id
attribute

Kobjects


Legend:
Kobject parent
Kobject type
Kset membership
block_subsys
.kset
ktype_block
hda
hdb
Request Queue
hda1
hda2
hda3
hdaN
hdb1
hdb2
hdb3
hdbN
Elevator
queue_ktype
as_ktype
ktype_part

Devices

- Cada *device* en el *ddm* es representado por un objeto *device*

Type	Field	Description
struct list_head	node	Pointers for the list of sibling devices
struct list_head	bus_list	Pointers for the list of devices on the same bus type
struct list_head	driver_list	Pointers for the driver's list of devices
struct list_head	children	Head of the list of children devices
struct device *	parent	Pointer to the parent device
struct kobject	kobj	Embedded kobject
char []	bus_id	Device position on the hosting bus
struct bus_type *	bus	Pointer to the hosting bus
struct device_driver *	driver	Pointer to the controlling device driver
void *	driver_data	Pointer to private data for the driver
void *	platform_data	Pointer to private data for legacy device drivers

- Los objetos *device* están listados en el subsistema `devices_subsys` asociado con el directorio `/sys/devices`
- La relación de parentesco entre los *kobjects* incrustados en los objetos *device* reflejan la jerarquía de los *devices* (también en `sysfs`)
- La función `device_register`() inserta el nuevo *device* en el *ddm* y crea un nuevo directorio bajo `/sys/devices`

Drivers

- Cada driver en el *ddm* es representado por un objeto `device_driver`

Type	Field	Description
`char *`	`name`	Name of the device driver
`struct bus_type *`	`bus`	Pointer to descriptor of the bus that hosts the supported devices
`struct semaphore`	`unload_sem`	Semaphore to forbid device driver unloading; it is released when the reference counter reaches zero
`struct kobject`	`kobj`	Embedded kobject
`struct list_head`	`devices`	Head of the list including all devices supported by the driver
struct module *	owner	Identifies the module that implements the device driver, if any (see Appendix B)
`int (*)(struct device *)`	`probe`	Method for probing a device (checking that it can be handled by the device driver)
`int (*)(struct device *)`	`remove`	Method invoked on a device when it is removed
`void (*)(struct device *)`	`shutdown`	Method invoked on a device when it is powered off (shut down)
`int (*)(struct device *, unsigned long, unsigned long)`	`suspend`	Method invoked on a device when it is put in low-power state
`int (*)(struct device *, unsigned long)`	`resume`	Method invoked on a device when it is put back in the normal state (full power)

Buses

- Cada tipo de bus soportado es representado en el *ddm* por un objeto `bus_type`

Type	Field	Description
`char *`	`name`	Name of the bus type
`struct subsystem`	`subsys`	Kobject subsystem associated with this bus type
`struct kset`	`drivers`	The set of kobjects of the drivers
`struct kset`	`devices`	The set of kobjects of the devices
struct bus_attribute *	bus_attrs	Pointer to the object including the bus attributes and the methods for exporting them to the *sysfs* filesystem
struct device_attribute *	dev_attrs	Pointer to the object including the device attributes and the methods for exporting them to the *sysfs* filesystem
struct driver_attribute *	drv_attrs	Pointer to the object including the device driver attributes and the methods for exporting them to the *sysfs* filesystem
`int (*)(struct device *, struct device_driver *)`	`match`	Method for checking whether a given driver supports a given device
`int (*)(struct device *, char **, int, char *, int)`	`hotplug`	Method invoked when a device is being registered
`int (*)(struct device *, unsigned long)`	`suspend`	Method for saving the hardware context state and changing the power level of a device
`int (*)(struct device *)`	`resume`	Method for changing the power level and restoring the hardware context of a device

- Cada objeto `bus_type` tiene un subsistema incrustado. (`/sys/bus`)
- El método `match()` se ejecuta cuando el kernel verifica que un *device* pueda ser manejado por un *device driver*
- El método `hotplug()` se ejecuta cuando un *device* se registra en el *ddm*

Classes

- Cada *class* es descrito por un objeto de tipo *class*
- Todos los objetos *class* pertenecen al subsistema `class_subsys` (`/sys/class`)
- Provee una abstracción de alto nivel sobre los dispositivos
- Su objetivo principal es proveer un método estándar para exportar las interfaces de los dispositivos hacia programas de *“user mode”*
- Cada objeto *class* contiene una lista de `class_device` que representa un único *device*
- Cada descriptor de `class_device` incrusta un *kobject* que tiene un atributo llamado *dev*. Este atributo contiene el *minor* y *major number* del *device*

Device files

- Los I/O devices son tratados como archivos especiales llamados *device files*
- Los *device files* pueden ser de dos tipos
  - Block
  - Character
- Los *inodos* de estos archivos no necesitan apuntar a bloques de datos. Deben contener un identificador del *device* correspondiente a un *device* file de tipo *character* o *block*
- Este identificador consiste en
  - El tipo de dispositivo
  - Un par de números (*major*, *minor*)
- El *major* identifica el tipo de dispositivo y el *minor* identifica un dispositivo dentro del grupo con el mismo *major*.

Device files

- La *syscall* `mknod()` se usa para crear los *device files*. Recibe el nombre del *device*, el tipo, el *major* y el *minor* como parámetros

Name	Type	Major	Minor	Description
/dev/fd0	block	2	0	Floppy disk
/dev/hda	block	3	0	First IDE disk
/dev/hda2	block	3	2	Second primary partition of first IDE disk
/dev/hdb	block	3	64	Second IDE disk
/dev/hdb3	block	3	67	Third primary partition of second IDE disk
/dev/ttyp0	char	3	0	Terminal
/dev/console	char	5	1	Console
/dev/lp1	char	6	1	Parallel printer
/dev/ttyS0	char	4	64	First serial port
/dev/rtc	char	10	135	Real-time clock
/dev/null	char	1	3	Null device (black hole)