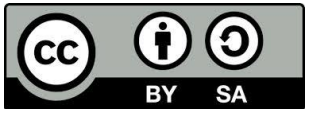
Drivers USB en Linux

LIN - Curso 2016-2017



J.C. Sáez
High Performance Computing Division
ArTeCs Group
Universidad Complutense de Madrid

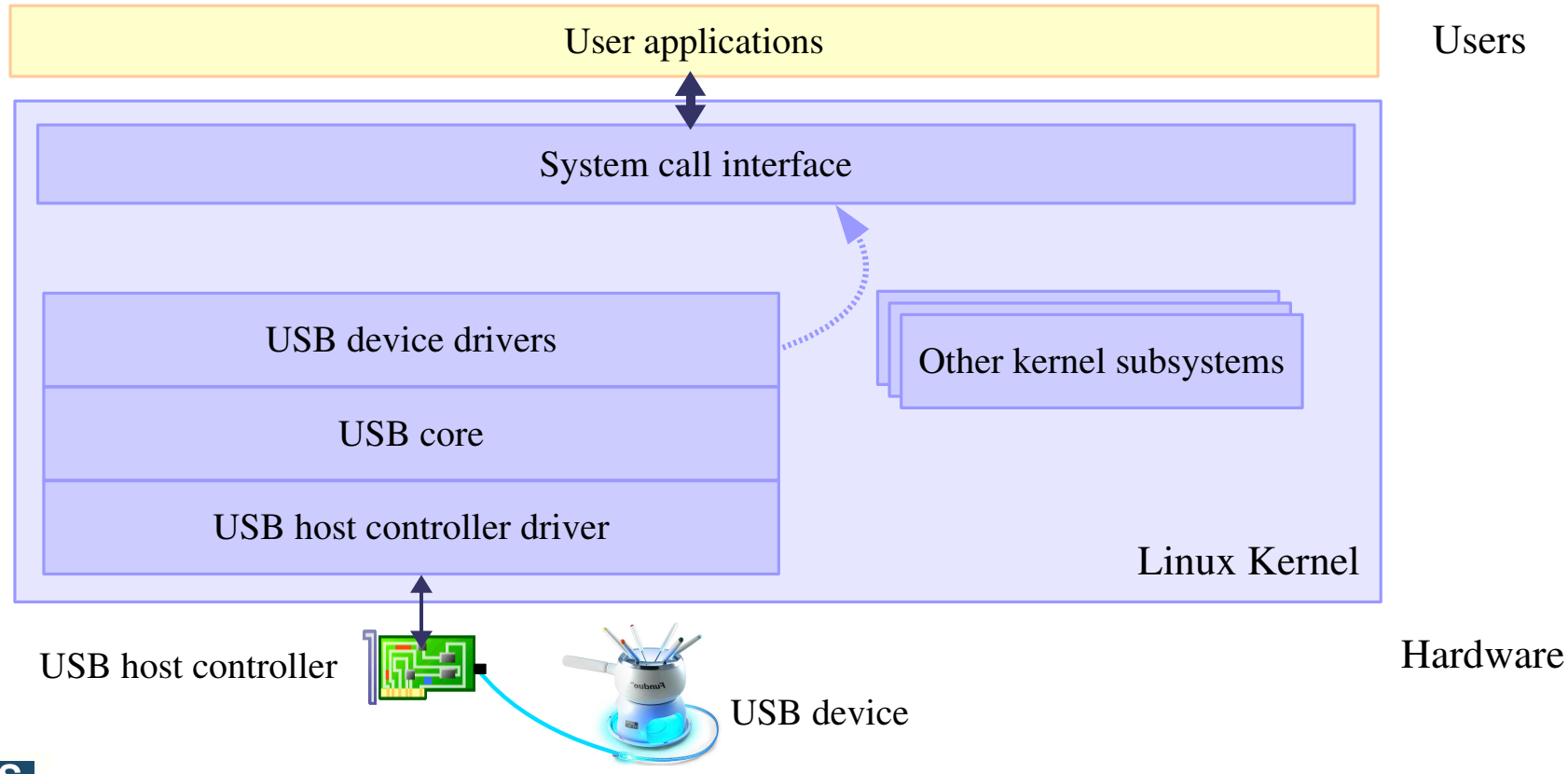
Contenido

Contenido

Universal Serial Bus

- USB (*Universal Serial Bus*) es una tecnología (tipo de conexión) que permite establecer la comunicación entre un computador (*host*) y uno o varios dispositivos periféricos
- **Objetivo**: Conexión de dispositivos de muy diversa velocidad
- No se implementa como un *bus*, sino un árbol de nodos conectados mediante conexiones punto a punto
  - Cada enlace está formado por 4 extremos de conexión (*ground*, *power*, 2x *signal*)
  - Cada dispositivo se conecta al *host* mediante un *puerto* o un *hub* USB

Dispositivos USB en Linux



Controladoras de Host USB - OHCI y UHCI

HDCs (*Host Control Device* (HCD))

- OHCI - Open Host Controller Interface
  - La implementación de Compaq se adoptó como estándar para USB 1.0 y 1.1 por parte de *USB Implementers Forum* (USB-IF)
  - Los dispositivos Firewire también la usan
- UHCI - Universal Host Controller Interface
  - Creada por Intel
  - Aquellos fabricantes que la usan tenían que pagar a Intel royalties por ella.

- La competición entre las dos interfaces de controlador obligaba a que los fabricantes tuvieran que testear los dispositivos para ambos tipos de controladores

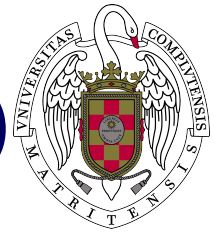
Controladoras de Host USB - EHCI

- Con la introducción de USB 2.0, USB-IF promovió la creación de un único estandar

EHCI - Enhanced Host Controller Interface

- Para USB 2.0
- Cada controlador EHCI implementa 4 tipos de HCD virtual para dar soporte a dispositivos USB de alta y baja velocidad
  - En chipsets Intel y VIA los HCDs virtuales son UHCI
  - Otros fabricantes de chipsets implementan HCDs virtuales OHCI

Controladoras de Host USB - XHCI (USB 3.x)



XHCI - Extensible Host Controller Interface

- Para USB 3.x y versiomes anteriores
  - USB 3.1 SuperSpeed+, USB 3.0 SuperSpeed, USB 2.0 Low-, Full-, and High-speed, USB 1.1 Low- and Full-speed
- Permite mayores velocidades de transferencia y mayor eficiencia energética que los antecesores

Velocidad de transferencia de dispositivos USB

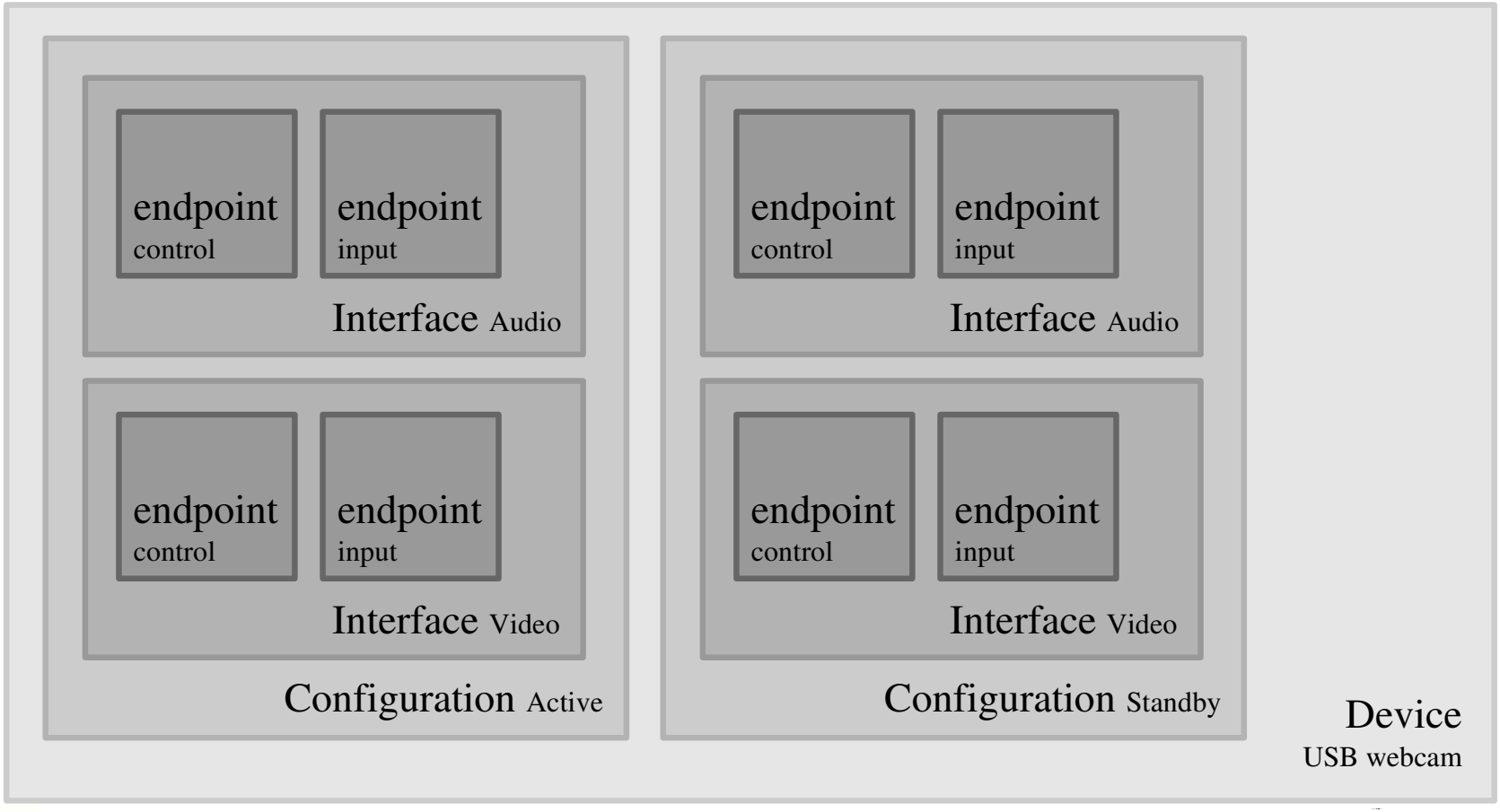
- **Low-speed**: hasta 1.5 Mbps
  - Desde USB 1.0
- **Full-speed**: hasta 12 Mbps
  - Desde USB 1.1
- **Hi-Speed**: hasta 480 Mbps
  - Desde USB 2.0
- **SuperSpeed** hasta 5 Gbps
  - Desde USB 3.0
- **SuperSpeed+** hasta 10 Gbps
  - Desde USB 3.1

Descriptores USB

- **Descriptores USB**: Elementos definidos en la especificación USB (Independientes del SO)

- **Device**: dispositivo físico conectado al bus USB
  - Ejemplo: Webcam USB con botones de control de volumen
- **Configurations:** Representan posibles estados del dispositivo
  - Ejemplo: Activo, En *Standby*, Inicialización
- **Interfaces:** Dispositivos lógicos dentro de un dispositivo USB
  - Ejemplo: Dispositivo de video, Dispositivo de audio, ...
- **Endpoints:** Canales de comunicación
  - Pueden ser `IN` (del dispositivo al host) o `OUT` (del host al dispositivo)

Jerarquía de descriptores USB



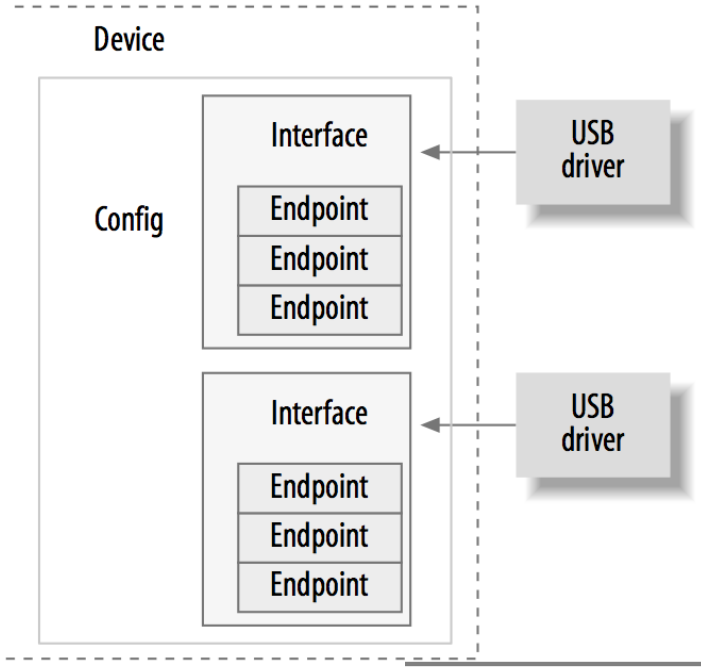
Tipos de *Endpoint*

USB *Endpoints*

- **Control**: Usado para configurar el dispositivo y recibir información
  - Transferencias pequeñas y simples
  - Todo dispositivo tiene al menos uno (control endpoint 0)
- **Interrupt**: Para transferir pequeñas cantidades de información a frecuencia fija
  - Ej.: para gestión de teclados y ratones USB
  - No son interrupciones HW. Requiere "encuesta" desde el host
- **Bulk**: para transferencias masivas de datos, pero esporádicas
  - Ej.: Impresoras, dispositivos de almacenamiento y de red
- **Isochronous**: para transferencias masivas de datos regulares
  - Ej.: transferencia de datos en tiempo real (audio, video, ...)

Interfaces de un dispositivo USB

- Cada interfaz encapsula una funcionalidad de alto nivel
  - Ejemplo para webcam USB: flujo de vídeo, flujo de audio, botones de control
- *Es necesario que haya un driver para cada interfaz USB*
- Cada interfaz puede tener sus propios parámetros de configuración



La estructura `usb_interface`

- En el kernel Linux, las interfaces de los dispositivos USB se representan mediante `struct usb_interface`
  - Definida en `<linux/usb.h>`
  - Esto es lo que USB core pasa a los drivers USB

Campos más relevantes

- `struct usb_host_interface *altsetting;`
  - Array de "modos alternativos" que se podrían seleccionar para esta interfaz
  - El campo `num_altsetting` indica el número de elementos del array
  - Desde cada modo se puede acceder a los *endpoints* (estructura usb_endpoint_descriptor)
    - `interface->alsetting[i]->endpoint[j]->desc`

La estructura `usb_interface`

Campos más relevantes (Cont.)

- `struct usb_host_interface *cur_altsetting;`
  - Modo activo
- `int minor;`
  - Minor number que esta interfaz tiene asociado.
  - Para drivers que usan `usb_register_dev()` (se describe a continuación).

Listando dispositivos USB

Terminal

```
kernel@debian:~$ usb-devices
T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=12  MxCh= 2
D:  Ver= 1.10 Cls=09(hub  ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0001 Rev=03.14
S:  Manufacturer=Linux 3.14.1.lin uhci_hcd
S:  Product=UHCI Host Controller
S:  SerialNumber=0000:02:00.0
C:  #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub

T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=0e0f ProdID=0003 Rev=01.03
S:  Manufacturer=VMware
S:  Product=VMware Virtual USB Mouse
C:  #Ifs= 1 Cfg#= 1 Atr=c0 MxPwr=0mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=usbhid

T:  Bus=01 Lev=02 Prnt=03 Port=01 Cnt=02 Dev#=  5 Spd=480 MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=090c ProdID=1000 Rev=11.00
S:  Manufacturer=USB
S:  Product=USB DISK
S:  SerialNumber=AA000000000000000073
C:  #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I:  If#= 0 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
```

USB Request Blocks - URBs

USB Request Blocks

- Cualquier comunicación entre el host y el dispositivo se realiza de forma asíncrona usando USB Request Blocks (URBs)
  - Un URB es un paquete de datos (mensaje)
  - En Linux se representa mediante `struct urb`
- En Linux hay dos APIs para enviar/recibir URBs:
  1. **Síncrona**: para transferir URBs de control sencillos
     - Los URBs no se gestionan explícitamente
     - Fácil de usar (llamadas bloqueantes)
  2. **Asíncrona**: usado en el resto de casos
     - Exige reservar memoria para URBs, inicializarlo, enviarlo y esperar a que acabe (*completion handler*)
     - Más versátil pero más compleja

Contenido

Dispositivo Blinkstick Strip

- Dispositivo USB que consta de 8 leds de colores
  - ¡¡¡LEDs muy brillantes!!!

Características

- Una sola interfaz USB (sólo requiere un driver)
- Un único *endpoint* de control (IN/OUT #0)
- Acepta URBs con mensajes de 8 bytes como máximo
  - Para establecer color de un led → enviar mensaje de 6 bytes

Conexión del dispositivo a la MV

- Tras conectar el dispositivo a la máquina virtual, consultamos la información sobre el mismo con `usb-devices`

```
Terminal
kernel@debian:~$ usb-devices
...
...
T:  Bus=01 Lev=02 Prnt=03 Port=01 Cnt=02 Dev#= 21 Spd=1.5 MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=20a0 ProdID=41e5 Rev=02.01
S:  Manufacturer=Agile Innovative Ltd
S:  Product=BlinkStick
S:  SerialNumber=BS001762-3.0
C:  #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=40mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=00 Prot=00 Driver=usbhid
```

- El dispositivo se expone como HID (*Human Interface Device*)
  - El driver genérico `usbhid` se hace cargo de él
  - Permite construir un driver en modo usuario con *libusb*

Conexión del dispositivo a la MV: dmesg

- Consultamos con dmesg los mensajes que muestra el USB core
  - "1-2.2" nos indica el puerto de conexión físico de nuestro dispositivo (host USB, hub, ...)

Terminal

```
kernel@debian:~$ dmesg | tail -n 7
[11402.964331] usb 1-2.2: new low-speed USB device number 21 using uhci_h
[11403.135337] usb 1-2.2: New USB device found, idVendor=20a0, idProduct=
[11403.135340] usb 1-2.2: New USB device strings: Mfr=1, Product=2, Seria
[11403.135341] usb 1-2.2: Product: BlinkStick
[11403.135343] usb 1-2.2: Manufacturer: Agile Innovative Ltd
[11403.135344] usb 1-2.2: SerialNumber: BS001762-3.0
[11404.446059] hid-generic 0003:20A0:41E5.0004: hiddev0,hidraw1: USB HID
```

Propiedades del dispositivo en sysfs

- Podemos consultar las propiedades del dispositivo en el directorio /sys/bus/usb/devices/<dir-usb>

Terminal

```
kernel@debian:~$ cd /sys/bus/usb/devices/1-2.2
kernel@debian:/sys/bus/usb/devices/1-2.2$ ls
1-2.2:1.0           bConfigurationValue  bmAttributes         bNumInterface
dev       ep_00         manufacturer  product    serial     urbnum
authorized          bDeviceClass         bMaxPacketSize0      busnum
devnum    idProduct     maxchild      quirks     speed      version
...
kernel@debian:/sys/bus/usb/devices/1-2.2$ cat bNumInterfaces
1
kernel@debian:/sys/bus/usb/devices/1-2.2$ cd ep_00/
kernel@debian:/sys/bus/usb/devices/1-2.2/ep_00$ ls
bEndpointAddress  bInterval  bLength  bmAttributes  direction  interval
kernel@debian:/sys/bus/usb/devices/1-2.2/ep_00$ cat type
Control
kernel@debian:/sys/bus/usb/devices/1-2.2/ep_00$ cat direction
both
kernel@debian:/sys/bus/usb/devices/1-2.2/ep_00$ cat wMaxPacketSize
0008
```

Contenido

Driver básico para Blinkstick Strip

- Como parte del material de la práctica 2 se proporciona un driver sencillo para el dispositivo Blinkstick

Características del driver (`blinkdrv.c`)

- Módulo del kernel que se registra como driver USB
- Se comporta como un driver de dispositivo de caracteres
  - Al conectar un dispositivo Blinkstick Strip, se crea automáticamente el fichero de dispositivo `/dev/usb/blinkstick0`
- El usuario puede cambiar el estado de los leds escribiendo en `/dev/usb/blinkstick0`
  - Procesamiento se lleva a cabo en función `blink_write()`
  - Funcionalidad muy limitada (a extender en la P2)

Cargando driver USB `blinkdrv.c` (1/4)

- Al conectar el dispositivo el driver genérico `usbhid` se hace cargo de él
  - Esto impide que lo podamos gestionar con otro driver
- Solución: hacer que `usbhid` ignore el dispositivo
  1. Descargar driver `usbhid` con `rmmod`
  2. Recargar driver `usbhid` pero introduciendo el dispositivo Blinkstick Strip en una *lista negra*
     - Posible gracias al parámetro `quirks` del módulo
     - Sintaxis valor: `<VENDOR_ID>:<DEVICE_ID>:<QUIRK_ID_COMMAND>`
     - QUIRK_ID_COMMAND=0x0004 → ignorar

Cargando driver USB `blinkdrv.c` (2/4)

Terminal

```
kernel@debian:~$ sudo rmmod usbhid
[sudo] password for kernel:
kernel@debian:~$ sudo modprobe usbhid quirks=0x20A0:0x41E5:0x0004
kernel@debian:~$ usb-devices  | tail
...
T:  Bus=01 Lev=02 Prnt=03 Port=01 Cnt=02 Dev#= 21 Spd=1.5 MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=20a0 ProdID=41e5 Rev=02.01
S:  Manufacturer=Agile Innovative Ltd
S:  Product=BlinkStick
S:  SerialNumber=BS001762-3.0
C:  #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=40mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=00 Prot=00 Driver=(none)
```

Cargando driver USB `blinkdrv.c` (3/4)

Terminal

```
kernel@debian:~/blinkdrv$ ls
blinkdrv.c  Makefile
kernel@debian:~/blinkdrv$ make
make -C /lib/modules/3.14.1.lin/build M=/home/kernel/blinkstick/src/blinkdrv module
make[1]: se ingresa al directorio `/usr/src/linux-headers-3.14.1.lin'
  CC [M]  /home/kernel/blinkdrv/blinkdrv.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/kernel/blinkdrv/blinkdrv.mod.o
  LD [M]  /home/kernel/blinkdrv/blinkdrv.ko
make[1]: se sale del directorio `/usr/src/linux-headers-3.14.1.lin'
kernel@debian:~/blinkdrv$ sudo insmod blinkdrv.ko
kernel@debian:~/blinkdrv$ stat /dev/usb/blinkstick0
  Fichero: «/dev/usb/blinkstick0»
  Tamaño: 0  Bloques: 0     Bloque E/S: 4096  fichero especial de caracteres
Dispositivo: 4h/4d  Nodo-i: 42060  Enlaces: 1  Tipo de dispositivo: b4,0
Acceso: (1660/crw-rw---T)  Uid: (    0/    root)   Gid: (    0/    root)
      Acceso: 2015-10-18 12:08:56.052838521 +0200
Modificación: 2015-10-18 12:08:56.052838521 +0200
      Cambio: 2015-10-18 12:08:56.052838521 +0200
    Creación: -
```

Cargando driver USB `blinkdrv.c` (4/4)

Terminal

```
kernel@debian:~/blinkdrv$ usb-devices  | tail
...
T:  Bus=01 Lev=02 Prnt=03 Port=01 Cnt=02 Dev#= 21 Spd=1.5 MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=20a0 ProdID=41e5 Rev=02.01
S:  Manufacturer=Agile Innovative Ltd
S:  Product=BlinkStick
S:  SerialNumber=BS001762-3.0
C:  #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=40mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=00 Prot=00 Driver=blinkstick
kernel@debian:~/blinkdrv$ echo lo que sea >  /dev/usb/blinkstick0
kernel@debian:~/blinkdrv$ echo  >  /dev/usb/blinkstick0
...
```

Al escribir en el fichero especial de caracteres, el driver cambia el color de todos los leds

Interfaz `usb_driver`

- Para que un módulo del kernel se comporte como un driver USB debe implementar la interfaz `usb_driver` (`<linux/usb.h>`)

```
struct usb_driver {
    const char *name;
    int (*probe) (struct usb_interface *intf,
            const struct usb_device_id *id);
    void (*disconnect) (struct usb_interface *intf);
    int (*unlocked_ioctl) (struct usb_interface *intf,
        unsigned int code,void *buf);
    int (*suspend) (struct usb_interface *intf, pm_message_t
         message);
    int (*resume) (struct usb_interface *intf);
    int (*reset_resume)(struct usb_interface *intf);
    int (*pre_reset)(struct usb_interface *intf);
    int (*post_reset)(struct usb_interface *intf);
    const struct usb_device_id *id_table;
    ...
};
```

Interfaz `usb_driver`

Campos más relevantes

- `name`: nombre del driver (arbitrario)
- `id_table`: tabla donde se especifican qué dispositivos son compatibles con el driver
  - Cada dispositivo compatible se especifica mediante su VENDOR_ID y su PRODUCT_ID
- `probe`: Función del driver que se ejecuta cuando el USB core detecta que se ha conectado un dispositivo compatible
  - El descriptor de la interfaz USB del dispositivo (abstracción del protocolo USB) se pasa como parámetro
- `disconnect`: Función del driver que se ejecuta cuando desconectamos un dispositivo gestionado por el driver

Registro de un driver USB: blinkdrv

```
static struct usb_driver blink_driver = {
    .name =    "blinkstick",
    .probe =   blink_probe,
    .disconnect = blink_disconnect,
    .id_table = blink_table,
};

/* Module initialization */
int blinkdrv_module_init(void)
{
   return usb_register(&blink_driver);
}

/* Module cleanup function */
void blinkdrv_module_cleanup(void)
{
  usb_deregister(&blink_driver);
}

module_init(blinkdrv_module_init);
module_exit(blinkdrv_module_cleanup);
```

Tabla de dispositivos compatibles: `blinkdrv`

blink_table

```
/* Define these values to match your devices */
#define BLINKSTICK_VENDOR_ID 0x20A0
#define BLINKSTICK_PRODUCT_ID 0x41E5

/* table of devices that work with this driver */
static const struct usb_device_id blink_table[] = {
    { USB_DEVICE(BLINKSTICK_VENDOR_ID, BLINKSTICK_PRODUCT_ID) },
    { }                 /* Terminating entry */
};
```

Cada vez que se conecte un dispositivo USB con vendor_id=020A0 y device_id=0x41e5 el USB core invocará la operación `probe()` del driver USB

Estado del dispositivo USB

- Para cada dispositivo USB gestionado el driver asocia un objeto con los atributos necesarios para poder trabajar con el dispositivo
  - Objeto modelado como una estructura específica del driver
- Campos habituales:
  - Puntero al descriptor del dispositivo (`struct usb_device`)
  - Puntero al descriptor de la interfaz USB que el driver gestiona (`struct usb_interface`)
  - Contador de referencias (`struct kref`)
    - Inicialización: `kref_init()`
    - Incremento/decremento: `kref_get()`, `kref_put()`

```
struct usb_blink {
    struct usb_device *udev;
    struct usb_interface *interface;
    struct kref   kref;
};
```

Inicialización de la estructura: blinkdrv


```
static int blink_probe(struct usb_interface *interface,
            const struct usb_device_id *id)
{
    struct usb_blink *dev;
    int retval = -ENOMEM;

    /* Allocate memory for a usb_blink structure. */
    dev = vmalloc(sizeof(struct usb_blink));

    if (!dev) {
        dev_err(&interface->dev, "Out of memory\n");
        goto error;
    }

    /* Initialize the various fields in the usb_blink structure */
    kref_init(&dev->kref);
    dev->udev = usb_get_dev(interface_to_usbdev(interface));
    dev->interface = interface;

    /* save our data pointer in this interface device */
    usb_set_intfdata(interface, dev);
...
..
}
```

Acceso a la estructura `usb_blink`

- **Problema**: *¿Cómo recuperar la estructura `usb_blink` desde la función de escritura `blink_write()`?*
  - Cada vez que un programa de usuario ejecuta la llamada al sistema write() sobre el dispositivo (p.ej., mediante `echo`) el driver debe realizar operaciones sobre el dispositivo
- Aproximación de `blinkdrv` (3 pasos)
  1. Cuando un programa abre el dispositivo, `blink_open()` recupera el puntero a la estructura a partir de la estructura `inode` pasada como parámetro
  2. El puntero a `usb_blink` se almacena en el campo `private_data` del parámetro `file` (`struct file*`), pasado a `blink_open()`
  3. El parámetro `file` se pasa también a `blink_write()` $\rightarrow$ se puede recuperar el puntero a `usb_blink` mediante el campo `private_data`

Major and minor numbers

- Recuerda: Cada fichero de dispositivo en Linux tiene asociado un par único (*major*,*minor*)
  - Dispositivos del mismo tipo tienen asociado mismo major pero distinto minor
  - Cada driver tiene asociado un *major* y un rango de *minor numbers* para ese *major*

Terminal

```
kernel@debian:~$ ls -l /dev/usb/blinkstick0
crw-rw--wT 1 root root 180, 0 oct 18 12:33 /dev/usb/blinkstick0
```

- Dos drivers podrían tener el mismo *major* asignado, pero trabajan con distintos rangos de *minor* numbers
  - Esto permite al kernel saber qué driver debe encargarse de gestionar cada dispositivo

Dispositivos USB: major and minor

- USB core asigna major numbers diferentes para distintas clases de dispositivos USB
  - `input`, `usb`, `ttyACM`, `tty_usb`,`hiddev`, ...
  - `cat /proc/devices`
  - `http://www.linux-usb.org/usb.devices.txt`
- Al conectar un dispositivo USB al computador, el dispositivo se expone al SO con un tipo predefinido
  - *Major number* para dispositivo está prefijado
- Los drivers USB pueden solicitar al kernel algún minor específico para su dispositivo con `usb_register_dev()`
  - Esta función se ocupa también de:
    1. Crear automáticamente el fichero de dispositivo en `/dev` (Udev)
    2. Asignar la interfaz de operaciones (`file_operations`) al fichero de dispositivo

Dispositivos USB: usb_register_dev()

usb_register_dev

```
extern int usb_register_dev(struct usb_interface *intf,
                            struct usb_class_driver *class_driver);
```

- `intf`: descriptor de la interfaz USB, parámetro de la operación `probe()`
- `class_driver`: estructura que ha de definir el driver

Dispositivos USB: usb_class_driver

usb_class_driver

```
struct usb_class_driver {
    char *name;
    char *(*devnode)(struct device *dev, umode_t *mode);
    const struct file_operations *fops;
    int minor_base;
};
```

- name: patrón para nombre de fichero de dispositivo
- devnode: función definida en el driver que retorna
  1. Ruta relativa (a /dev) donde se creará el fichero de dispositivo
  2. Permisos del fichero de dispositivo (parámetro mode)
- minor_base: El comienzo del rango de minors asignados para este driver
  - El major viene prefijado (depende de tipo de dispositivo)

Dispositivos USB: usb_class_driver

usb_class_driver (Cont.)

```
struct usb_class_driver {
    char *name;
    char *(*devnode)(struct device *dev, umode_t *mode);
    const struct file_operations *fops;
    int minor_base;
};
```

- file_operations: puntero a las operaciones que el driver ejecutará cuando algún programa acceda al dispositivo
  - Misma interfaz que en /proc
  - El driver debe instanciar la estructura file_operations e implementar las operaciones

usb_class_driver en blinkdrv

```
#define USB_BLINK_MINOR_BASE 0

static const struct file_operations blink_fops = {
    .owner =   THIS_MODULE,
    .write =   blink_write, /* write() syscall */
    .open =    blink_open, /* open() syscall */
    .release = blink_release, /* close() syscall */
};

char* set_device_permissions(struct device *dev, umode_t *mode)
{
    if (mode)
        (*mode)=0666; /* RW permissions */
    return kasprintf(GFP_KERNEL, "usb/%s", dev_name(dev));
}

static struct usb_class_driver blink_class = {
    .name =    "blinkstick%d",
    .devnode=  set_device_permissions,
    .fops =    &blink_fops,
    .minor_base = USB_BLINK_MINOR_BASE,
};
```

Registrando usb_class_driver: blinkdrv


```
static int blink_probe(struct usb_interface *interface,
            const struct usb_device_id *id)
{
    struct usb_blink *dev;
    /* Initialization of the usb_blink structure */
    ...

    /* we can register the device now, as it is ready */
    retval = usb_register_dev(interface, &blink_class);
    if (retval) {
        /* something prevented us from registering this driver */
        dev_err(&interface->dev,
            "Not able to get a minor for this device.\n");
        usb_set_intfdata(interface, NULL);
        goto error;
    }

    /* let the user know what node this device is now attached to */
    dev_info(&interface->dev,
         "Blinkstick device now attached to blinkstick-%d",
         interface->minor);
    return 0;
    ...
}
```

Modificar el estado de los leds: `blinkdrv`

- Para establecer el color de un led es preciso enviar un URB de control al dispositivo USB usando el *endpoint* 0
- El URB debe llevar un mensaje de 6 bytes con la siguiente estructura:

Byte	Contenido
0	`'\x05'` (wValue)
1	0 (wIndex)
2	Número de led que se quiere modificar (0..7)
3	Componente roja del color (0..FF)
4	Componente verde del color (0..FF)
5	Componente azul del color (0..FF)

Envío/recepción URBs de forma síncrona

```
int usb_control_msg(struct usb_device *dev, unsigned int pipe,
    __u8 request, __u8 requesttype, __u16 value, __u16 index,
    void *data, __u16 size, int timeout);
```

Parámetros

- `dev`: Descriptor del dispositivo
- `pipe`: Canal para la comunicación. Se construye con `usb_sndctrlpipe()`.
- `data`: puntero al buffer que contiene el mensaje
- `size`: tamaño en bytes del mensaje
- `timeout`: tiempo (en ms) que puede durar la transacción
  - Si >0, la transacción se cancela si se supera el tiempo
  - Si 0, no hay límite de tiempo.

Envío/recepción URBs de forma síncrona

```
int usb_control_msg(struct usb_device *dev, unsigned int pipe,
    __u8 request, __u8 requesttype, __u16 value, __u16 index,
    void *data, __u16 size, int timeout);
```

Resto de Parámetros (Especificación USB 2.0 - Tabla 9.2)

- `request`: Tipo de petición de control
  - Ej.: `USB_REQ_SET_CONFIGURATION`
  - Constantes definidas en `<uapi/linux/usb/ch9.h>`
- `requestType`: Máscara de bits que contienen las características de la petición (dirección, tipo de dispositivo y receptor del mensaje)
  - Ej.: `USB_DIR_OUT| USB_TYPE_CLASS | USB_RECIP_DEVICE`
  - Constantes definidas en `<uapi/linux/usb/ch9.h>`
- `value` e `index`: campos que dependen de la petición y del dispositivo

Establecer color de leds en `blink_write()`

```
#define NR_BYTES_BLINK_MSG 6
#define NR_LEDS 8
static ssize_t blink_write(struct file *file, const char *user_buffer,
            size_t len, loff_t *off) {
   struct usb_blink *dev=file->private_data;
   int i=0;
   unsigned char message[NR_BYTES_BLINK_MSG];

   /* Fill up the message accordingly */
   ...
   for (i=0;i<NR_LEDS;i++){
      message[2]=i; /* Change Led number in message */

      retval=usb_control_msg(dev->udev,
          usb_sndctrlpipe(dev->udev,00), /* Specify endpoint #0 */
          USB_REQ_SET_CONFIGURATION,
          USB_DIR_OUT| USB_TYPE_CLASS | USB_RECIP_DEVICE,
          0x5,  /* wValue */
          0,    /* wIndex=Endpoint # */
          message, /* Pointer to the message */
          NR_BYTES_BLINK_MSG, /* message's size in bytes */
          0);
      ...
   }
...
```

Referencias

- M. Opdenacker. *Linux USB Drivers*. Sept, 2009.
  - `http://free-electrons.com/`
- Enlaces sobre USB:
  - Host controller interface (USB, Firewire)
  - xHCI for Universal Serial Bus: Specification
  - Documentación sobre URBs en las fuentes del kernel
  - Dispositivos de interfaz humana USB
- Linux Device Drivers (`https://lwn.net/Kernel/LDD3/`)
  - Cap. 13 *"USB Drivers"*
  - Cap. 14 *"Linux Device Model"*

Referencias

Blinkstick

- https://www.blinkstick.com
- https://www.blinkstick.com/help/overview
- https://www.blinkstick.com/products/blinkstick-strip
- https://www.blinkstick.com/help/schematics

Licencia

LIN - Drivers USB en Linux
Versión 0.5

Este documento (o uno muy similar) está disponible en
`https://cv4.ucm.es/moodle/course/view.php?id=75410`