Universidad Nacional Autónoma de México

Facultad de Ingeniería

Sistemas Operativos

Drivers

Meneses Navarro Erick Sebastian
Mex Lozano Aerin Musette

¿Qué son los drivers?

Un controlador o driver es un componente intermedio entre el hardware y el software que permite al sistema operativo comunicarse con los diferentes componentes del ordenador y pueda reconocer de forma correcta los componentes que se agregan al computador.

Son indispensables, ya que muchos dispositivos necesitan los controladores como por ejemplo, para el funcionamiento de impresoras, tarjetas gráficas, de audio, red, video, USB,etc, sin ellos no podríamos jugar, imprimir documentos o escuchar música.

Sin embargo, es importante destacar que en algunos casos, con un solo driver pueden funcionar multitud de unidades diferentes. Por ejemplo, un pendrive usará el controlador USB para funcionar, por lo que aunque cambies de pendrive no será necesario usar otro driver.

Los drivers están compuestos por líneas de código con funciones específicas dirigidas a componentes particulares, dichos programas se encargan de llevar un mensaje al sistema operativo para que este lo reconozca y se comunique con el dispositivo, dichas líneas de código generalmente son diseñados por la misma compañía que haya desarrollado el hardware, sin embargo, no es necesario que los drivers sean escritos por la misma empresa que haya diseñado el dispositivo, ya que, un dispositivo es diseñado según el estándar de hardware publicado; en Linux , por ejemplo, para una usb si no los publican entonces los desarrolladores se verán en la necesidad de utilizar ingeniería inversa.

¿Cómo funcionan?

Como ya se dijo, los drivers son los encargados de decirle al sistema operativo como usar el hardware que este tiene a su disposición, pero ¿que hace exactamente el driver para que el sistema operativo logre esto?

Dijimos también, que los drivers son un conjunto de líneas de código con instrucciones específicas, entonces, si tu quieres desarrollar un driver, ¿Como empezar?

Tenemos que tener ciertos requisitos previos para poder empezar a desarrollar, por ejemplo, para Linux, pues entonces necesitamos tener un profundo conocimiento en el lenguaje de programación C, cosas como el uso de punteros, funciones de manipulación de bits, etc. y también se necesita conocer cómo es que internamente funcionan las microcomputadoras, direccionamiento de memoria, interrupciones, todo esto, por tanto, se debe estar familiarizado con programación con microprocesadores.

Es importante distinguir entre lo que es el espacio del kernel y el del usuario, donde este primero forma un puente entre el usuario final / programador y el hardware, cualquier subrutina o función que forma parte del kernel, como los drivers, están en el espacio del kernel.

El espacio del usuario son los programas que utiliza el usuario final, como UNIX por dar un ejemplo, dichas aplicaciones necesitan interactuar con el hardware del sistema, pero no directamente si no con las funciones compatibles con el kernel.

¿Funciones? Bueno, pues tenemos funciones de interfaz entre el espacio kernel y el espacio del usuario; el espacio de kernel de Linux proporciona un serie de subrutinas en el espacio de usuario que permite al programador de usuario final interacciones de bajo nivel directamente con el hardware y permitir la transferencia de información desde el kernel al espacio de usuario.

Por cada función en el espacio de usuario, el que permite el uso de archivos, existe su equivalente en el espacio del núcleo, el que hace que se transfieran los datos del kernel al usuario y viceversa.

También hay funciones en el espacio del kernel que controlan el dispositivo o intercambian información entre el kernel y el hardware.

Evento	Función del usuario	Función del kernel
Carga del módulo	insmod	init_module
Abrir dispositivo	fopen	file operations: open
Leer dispositivo	fread	file operations: read
Escribir dispositivo	fwrite	file operations: write
Cerrar dispositivo	fclose	file operations: release
Quitar módulo	rmmod	cleanup_module

En sistemas Linux, los propios dispositivos son vistos desde el punto de vista del usuario como ficheros, es entonces que este diálogo se hace a través de las funciones o subrutinas de lectura y escritura de ficheros.
Para ponernos en contexto, vamos a usar un ejemplo, donde, se crea un código base para dispositivos USB

Para empezar, en linux, es posible conectar una gran variedad de dispositivos USB, pero siempre existirá alguno que no esté en la lista de dispositivos compatibles, por

lo que toca crearlo, pero, debido a que cada protocolo crea un nuevo driver, se crea un código base, un “esqueleto” que se pueda modificar a conveniencia toda esta información se pasa a la estructura de la usb y es declarada como

```
static struct usb_driver skel_driver = {
        .name        = "skeleton",
        .probe       = skel_probe,
        .disconnect  = skel_disconnect,
        .suspend     = skel_suspend,
        .resume      = skel_resume,
        .pre_reset   = skel_pre_reset,
        .post_reset  = skel_post_reset,
        .id_table    = skel_table,
        .supports_autosuspend = 1,
};
```

Este esqueleto es para registrar el dispositivo en el subsistema de la usb en la estructura usb_driver

Las variables probe y disconnect son apuntadores que son llamados cuando el dispositivo coincide con la información brindada en la variable id_table ya sea usada o removida.

Las demás variables pueden ser opcionales, ya que este tipo de drivers se registran a sí mismos en otro subsistema en el kernel y cualquier otra interacción en el espacio usuario es brindada en esta interfaz, pero para nuestro caso de la creación del driver “esqueleto” si, puesto que, el subsistema de la usb brinda una forma de registrar una mínima cantidad de número de dispositivos y un conjunto de las funciones de archivos que entablan la interacción en el espacio usuario.

Para registrar el driver es necesario llamar a la funcion usb_registrer ubicada en la función inicial del driver.

```
static int __init usb_skel_init(void)
{
        int result;

        /* register this driver with the USB subsystem */
        result = usb_register(&skel_driver);
        if (result < 0) {
                pr_err("usb_register failed for the %s driver. Error number %d\n",
                       skel_driver.name, result);
                return -1;
        }

        return 0;
}
module_init(usb_skel_init);
```

Podemos observar que ya tenemos la carga del módulo gracias al función module_init.

Se requiere una función para quitar el módulo y cancelar el registro con el subsistema de la usb, llamando a module_exit

```
static void __exit usb_skel_exit(void)
{
        /* deregister this driver with the USB subsystem */
        usb_deregister(&skel_driver);
}
module_exit(usb_skel_exit);
```

Cuando enchufamos la usb hay que activar el linux-hotplug system para cargar el driver automáticamente, lo mostrado a continuación indica que este módulo admite un solo dispositivo con un proveedor y id en específico del producto.

```
/* table of devices that work with this driver */
static struct usb_device_id skel_table [] = {
        { USB_DEVICE(USB_SKEL_VENDOR_ID, USB_SKEL_PRODUCT_ID) },
        { }                     /* Terminating entry */
};
MODULE_DEVICE_TABLE (usb, skel_table);
```

Ahora que ya tenemos la carga y descarga del módulo del dispositivo USB veremos cómo es que debemos desarrollar la parte del funcionamiento de este.

Cuando conectamos la USB necesita ser verificado para que coincida con el ID registrado, la usb_strcture, número de interfaz e interfaz ID se pasan en la siguiente función

```
static int skel_probe(struct usb_interface *interface,
    const struct usb_device_id *id)
```

Si lo acepta entonces retornará un cero, si no entonces ocurrirá un error durante la inicialización retornando en la función “probe” un error.

Con esto, se determinan los puntos finales marcados como “bulk-in” y “bulk-out”, se crean los buffer para tener la información que se reciba o se envíe al dispositivo y un “urb”(USB request block) para inicializar el dispositivo.

Al momento de desconectar la USB, se llama a la función encargada y se tiene que limpiar toda la información privada usada en el dispositivo, además de acabar la ejecución de cualquier “urb” pendiente.

Ahora que el dispositivo está conectado al sistema y el driver está enlazado al dispositivo, cualquiera de las funciones del esqueleto se pasaron al subsistema usb y serán llamadas desde el programa usuario tratando de hablar con el dispositivo.

La función de apertura del dispositivo será llamada, mientras el programa trata de abrir el dispositivo mediante E/S esto con las file operations, mas en especifico fopen, después de que se realice las demás funciones como la de lectura y escritura serán llamadas para recibir y enviar los datos hacia el dispositivo, en la función recibiremos un apuntador que tenga los datos que el usuario desea mandar al dispositivo así como el tamaño de los mismos a través de la información que nos brinda el urb generado.

```
/* we can only write as much as 1 urb will hold */
size_t writesize = min_t(size_t, count, MAX_TRANSFER);

/* copy the data from user space into our urb */
copy_from_user(buf, user_buffer, writesize);

/* set up our urb */
usb_fill_bulk_urb(urb,
                  dev->udev,
                  usb_sndbulkpipe(dev->udev, dev->bulk_out_endpointAddr),
                  buf,
                  writesize,
                  skel_write_bulk_callback,
                  dev);

/* send the data out the bulk port */
retval = usb_submit_urb(urb, GFP_KERNEL);
if (retval) {
        dev_err(&dev->interface->dev,
            "%s - failed submitting write urb, error %d\n",
            __func__, retval);
}
```

Entonces se copian los datos del espacio usuario al espacio kernel al apuntar el urb hacia los datos y subir este al subsistema USB.

Pasando a la función de lectura este no utiliza la urb para pasar la información si no que llama a la funcion usb_bulk_msg() para mandar y recibir la información del dispositivo al driver, al darle al buffer cualquier dato recibido con un tiempo límite, si este expira la función retornará un mensaje de error.

```
/* do an immediate bulk read to get data from the device */
retval = usb_bulk_msg (skel->dev,
                       usb_rcvbulkpipe (skel->dev,
                       skel->bulk_in_endpointAddr),
                       skel->bulk_in_buffer,
                       skel->bulk_in_size,
                       &count, 5000);
/* if the read was successful, copy the data to user space */
if (!retval) {
        if (copy_to_user (buffer, skel->bulk_in_buffer, count))
                retval = -EFAULT;
        else
                retval = count;
}
```

Cuando el programa usuario deja de utilizar el driver para comunicarse con el dispositivo se usa la función release() decrementando el valor de conteo de uso privado esperando peticiones potenciales de escritura.

```
/* decrement our usage count for the device */
--skel->open_count;
```

Ahora, un detalle que se debe tomar a consideración es cuando, de manera repentina, se extrae la usb en cualquier momento del sistema, aún cuando el programa se esté comunicando con él, es por eso que es necesario poder detener cualquier lectura y escritura actuales e informando a los programas de espacio usuario que el dispositivo no está.

```
static inline void skel_delete (struct usb_skel *dev)
{
    kfree (dev->bulk_in_buffer);
    if (dev->bulk_out_buffer != NULL)
        usb_free_coherent (dev->udev, dev->bulk_out_size,
            dev->bulk_out_buffer,
            dev->write_urb->transfer_dma);
    usb_free_urb (dev->write_urb);
    kfree (dev);
}
```

Por cada función de lectura, escritura, liberación y cualquier otra función que requiera al dispositivo tiene una bandera que verificará si está presente o no, en caso de que no esté entonces se retornará un mensaje de error en el espacio usuario.
Pero para el momento se extrae de forma correcta al llamar a la función release() (cuando expulsas el dispositivo USB) ya se reconoce que no hay un dispositivo y se hace la respectiva limpieza, siempre que no haya archivos abiertos en el dispositivo.

Referencias

- Aviviano. (2023, 8 marzo). *¿Qué es un controlador? - Windows drivers*. Microsoft Learn. https://learn.microsoft.com/es-es/windows-hardware/drivers/gettingstarted/what-is-a-driver-
- Ferri-Benedetti, F. (2021, 25 junio). ¿Qué son los drivers o controladores? *Softonic*. https://www.softonic.com/articulos/que-son-los-drivers-controladores
- *Writing device drivers in Linux: A brief tutorial*. (s. f.). http://freesoftwaremagazine.com/articles/drivers_linux/
- *Writing USB Device Drivers — The Linux Kernel documentation*. (s. f.). https://www.kernel.org/doc/html/latest/driver-api/usb/writing_usb_driver.html