Características de los gestores de dispositivos en Linux

Tipos de drivers

En primer lugar, recordaremos que en Linux los dispositivos (y por extensión, los gestores de dispositivos) se clasifican en tres categorías:

1. de **caracteres**: son dispositivos simples que utilizan la entrada/salida por encuesta o por interrupciones. A estos dispositivos puede accederse como si fueran una "sucesión de bytes". Son parecidos a un fichero, sólo que en el fichero podemos movernos hacia adelante y hacia atrás, mientras que los dispositivos de caracteres suelen ser meros *canales* de bytes a los que se accede secuencialmente (aunque hay excepciones). Dispositivos de caracteres en Linux son, por ejemplo, la consola de texto */dev/console* o los puertos serie como */dev/ttyS0*. Estos son los drivers que se pueden "instalar" en MaRTE gracias al entorno preparado para ello y de los únicos que nos vamos a ocupar en este proyecto.

2. de **bloques**: son dispositivos que pueden albergar un sistema de ficheros (como, por ejemplo, un disco). Suelen ser, por tanto, dispositivos que almacenan cantidades masivas de datos y que sólo pueden realizar operaciones de entrada/salida en las que se transfieren uno o más *bloques* de datos, normalmente de 512 bytes (o una potencia de dos mayor) de longitud. Por tanto, la mejor opción para realizar la entrada/salida en este caso es mediante acceso directo a memoria. Estos drivers tienen una interfaz con el kernel radicalmente distinta a la que tienen los drivers de caracteres.

3. de **red**: o *network interfaces*, donde la entrada/salida se realiza por dispositivos especializados de comunicaciones, es decir, dispositivos capaces de intercambiar datos con otros computadores (hosts) en una red. De nuevo en este caso el interfaz suele ser un dispositivo hardware, pero también puede ser software, como el interfaz de loopback (que reenvía los paquetes hacia el propio host emisor). En todo caso, el objetivo de un driver de red es la gestión del envío y recepción de paquetes de datos, dirigido por el subsistema de red del kernel del sistema.

Uso de los drivers

Un objetivo de los drivers es ofrecer una forma general de uso. Por ello, en Linux tienen una interfaz bien definida con el kernel, mediante unos *puntos de entrada* (o *métodos*) que en realidad son funciones C y que se muestran en la tabla 1. No es necesario implementar todos los puntos de entrada posibles, sino sólo aquellos que se necesiten para el dispositivo concreto. Existen muchos más puntos de entrada, que pueden consultarse por ejemplo en el capítulo

open	abrir el dispositivo para usarlo
release	cerrar el dispositivo
read	leer un número de bytes del dispositivo
write	escribir un número de bytes en el dispositivo
ioctl	se utiliza para el control flexible del dispositivo

Tabla 1: Puntos de entrada más frecuentes en drivers Linux

3 del libro *Linux Device Drivers* [6]. Los puntos de entrada permiten que las aplicaciones utilicen las capacidades del driver, pero lo que hacen las aplicaciones es llamar al kernel mediante una *interfaz de llamadas al sistema* para que después el kernel llame al punto de entrada adecuado del driver. Las llamadas al sistema tienen por lo general un nombre muy similar al punto de entrada del driver.

Archivos de dispositivo

En Linux sabemos que todos los dispositivos son tratados por el sistema como ficheros. Los *archivos de dispositivo* ("device nodes" o "device files") son ficheros especiales que constituyen la interfaz usuario–dispositivo. Residen por lo general en `/dev` y se crean con `mknod`. Una vez abierto el archivo mediante "open" (especificando el *path* del archivo de dispositivo), las llamadas al sistema especifican a qué dispositivo se refieren gracias a un *descriptor de fichero*, que es un número entero que el kernel asocia a cada fichero abierto. Por ejemplo, la llamada al sistema *read* tiene la especificación siguiente:

```
#include <sys/types.h>
size_t read(int fildes,void *buf,size_t nbyte);
```

Donde `fildes` es el descriptor de fichero, `buf` la variable donde se guardará el dato leído y `nbyte` el número de bytes que se desean leer. La función devuelve el número de bytes leídos.

Los archivos de dispositivo se identifican mediante los denominados *números mayores y menores*. Cada archivo de dispositivo tendrá asignado un número mayor y un número menor, que pueden consultarse simplemente ejecutando `$ls -l /dev` en una ventana de shell.

Puede entenderse ahora mejor la utilización del término *"métodos"* para los puntos de entrada del driver (ya que, siguiendo la filosofía de programación orientada a objetos (OOP), en el caso Linux se consideraría que el archivo de dispositivo es el "objeto").

Sincronización y acceso mutuamente exclusivo

El acceso mutuamente exclusivo se suele conseguir, en los sistemas Unix, habilitando e inhabilitando interrupciones mediante las rutinas del kernel cli() y sti():

```
cli()
   Operaciones en la sección crítica
sti()
```

Linux utiliza también **semáforos**, **mutexes**, **spinlocks** y **operaciones atómicas** (con variables atómicas) para la exclusión mutua. Los semáforos también sirven para la sincronización, pero se dispone de herramientas más sofisticadas como **completions** y las **colas de**

procesos ("wait queues"). Con estas últimas pueden dormirse procesos (con las funciones de tipo *wait_event*) y posteriormente encontrarse para ser despertados (con las funciones de tipo *wake_up*). Todas estas cuestiones revisten cierta complejidad y pueden consultarse en los capítulos 5 y 6 de *Linux Device Drivers* [6].

Espacio de kernel y espacio de usuario

Una diferencia importante entre Linux y MaRTE es que en el primer sistema existe una separación entre el **espacio de kernel** y el **espacio de usuario**. Así, una aplicación de usuario *no tiene permiso* para acceder libremente a los dispositivos de entrada/salida, sino que debe hacerlo siempre a través de los drivers. En Linux los drivers son módulos software que pueden *cargarse* y *descargarse* dinámicamente del kernel mientras el sistema está corriendo, sin necesidad de reiniciar la máquina. Eso sí, dependiendo de la configuración del kernel y del driver a instalar, puede ser necesario *recompilar el kernel*. El módulo debe tener al menos dos operaciones:

1. init_module: para instalar el módulo
2. cleanup_module: para desinstalarlo

Como parte del kernel, los drivers tienen los privilegios necesarios para gestionar los dispositivos, pero deben emplear funciones especiales del kernel (distintas de las funciones de librería o las llamadas al sistema que se emplean en el código de las aplicaciones de usuario). Por ejemplo, se emplea la función `printk` en vez de `printf`; la función `printk` tiene entre sus argumentos la prioridad con que quiere que se muestre el mensaje en pantalla:

```
int init_module(void) {
  printk("<1>Hola mundo\n");
  return 0;}

void cleanup_module(void) {
  printk("<1>Adios mundo cruel\n");}
```

A menor número, más alta prioridad. De esta manera, si la prioridad es <1> conseguimos que el mensaje aparezca efectivamente en la pantalla y no se quede simplemente guardado en los ficheros de mensajes del kernel.

Existen, además, funciones que sirven para el intercambio de información entre el espacio de kernel y el espacio de usuario, como por ejemplo `put_user(datum, ptr)`. Estas funciones suelen usarse, por ejemplo, en el método *ioctl* para mover información de forma segura hacia el espacio de usuario. Para más información, consultar el capítulo 6 del libro *Linux Device Drivers*[6].

Seguridad

Son precisamente los privilegios conferidos a los drivers los que los convierten en fuentes potenciales de inseguridad para un sistema, y por eso la fiabilidad en su diseño resulta crítica. Muchos drivers mal diseñados resultan ser una puerta de entrada para usuarios malintencionados.