# Práctica II

# Character Device Drivers

**Implementación de Manejadores de Dispositivos**

**Maestría en Sistemas Embebidos**

## Año 2024

**Autor**

## Mg. Ing. Pablo Slavkin

## Mg. Ing. Hanes Nahuel Sciarrone

# Tabla de contenido

# Registro de cambios

| Revisión | Cambios realizados | Fecha |
|---|---|---|
| 1.0 | Creación del documento | 05/11/2021 |
| 1.1 | Actualización de tabla de contenidos | 08/11/21 |
| 1.2 | Correcciones ortográficas varias | 09/11/21 |
| 1.3 | Modificación de encabezado | 16/02/2024 |

# Escribiendo un módulo “hola mundo char”

En general los sistemas linux utilizaban un método de creación de dispositivos estático, donde una gran cantidad de nodos de dispositivos eran creados dentro de **/dev**, y esto no necesariamente correspondía a dispositivos de hardware que realmente existieran. Este proceso era efectuado por un script llamado MAKEDEV, el cual contiene una cierta cantidad de llamadas al comando **mknod**. Las llamadas eran efectuadas con los números major & minor para cada posible dispositivo existente en el planeta.

Claramente esta técnica dejó de utilizarse. Hoy en día se debe crear la entrada manualmente y luego asociarla con el dispositivo, según se muestra en el siguiente comando:

```
$ mknod /dev/mydev c 202 0
```

Tomar en cuenta que los números seleccionados son arbitrarios y deben ser modificados si en el sistema existe ya algún dispositivo con esta combinación de números.

La creación de un nodo manual (estáticamente) es meramente a fines educativos, esta no es la manera utilizada en la práctica dado que hay métodos mejores utilizando el filesystem **devtmpfs** y el **miscellaneus framework**.

En los pasos siguientes se utilizarán las syscalls **open()** e **ioctl()** en la aplicación espacio usuario y asimismo serán escritas las callbacks correspondientes en el driver, haciendo posible la comunicación entre el espacio usuario y el espacio kernel.

## Registro y baja de character devices

El registro y la baja de un char device se efectúa por medio de la especificación de sus números major & minor. El tipo de datos **dev_t** se utiliza para mantener los identificadores de un dispositivo (ambos números mencionados) y puede ser obtenido mediante la macro MKDEV.

Para la asignación estática y liberación de los identificadores de dispositivos, se utilizan las funciones **register_chrdev_region()** y **unregister_chrdev_region()**.

```
int register_chrdev_region(dev_t first, unsigned int count, char *name);

void unregister_chrdev_region(dev_t first, unsigned int count);
```

Se puede reemplazar la utilización de **register_chrdev_region()** por una versión de adquisición de número major dinámica, que es **alloc_chrdev_region()**. Esta función tiene la facilidad de buscar de forma dinámica un número major libre, pero tiene la dificultad de que de alguna manera al registrar el dispositivo, se debe crear el nodo con el número major encontrado. Para esta práctica utilizaremos **register_chrdev_region()** a fines de simplicidad.

Se requiere al lector que descargue el archivo **holamundo_char_driver.c**, compile el módulo y ejecute los siguientes comandos, instalando el módulo y creando el nodo correspondiente al dispositivo registrado:

```
$ insmod holamundo_char_driver.ko

$ cat /proc/devices
```

La salida del último comando muestra que el dispositivo está correctamente registrado, pero aún no existe un nodo en el sistema al cual asociarlo, y esto puede verificarse ejecutando:

```
$ ls -l /dev
```

Cl caramente se ve que el nodo correspondiente no existe. Se debe crear mediante:

```
$ mknod /dev/mydev c 202 0
```

Si nuevamente se ejecuta **ls** se nota que el nodo ahora existe:

```
$ ls -l /dev
```

Habiendo sido verificado que el nodo existe y que el módulo fue cargado de manera correcta en el sistema, se puede correr la aplicación **ioctl_test** espacio usuario para verificar el funcionamiento del módulo. El código fuente **ioctl_test.c** debe ser descargado y compilado para la arquitectura actual.

```
$ ./ioctl_test
```

```
$ rmmod holamundo_char_driver
```

## Creación de device files con devtmpfs

Antes del kernel 2.6.32, los sistemas básicos de linux requerían que los device files fueran creados a mano utilizando el comando mknod, como se hizo en la práctica anterior. La coherencia entre los device files y los dispositivos manejados por el kernel era responsabilidad exclusiva del desarrollador. Luego de la llegada de la versión estable mencionada, un nuevo filesystem virtual llamado *sysfs* fue agregado. El trabajo de *sysfs* es exportar una vista del hardware del sistema a los procesos de espacio usuario.

Sysfs sabe sobre los dispositivos presentes en un sistema porque los drivers que son compilados en el kernel directamente registran sus objetos con un sysfs al momento de ser detectados por el kernel. En el caso de drivers compilados como módulos, este registro sucede al momento de ser cargados. Una vez que el *sysfs* es montado en el directorio */sys*, los datos que registran los drivers está disponible en espacio usuario para que **udev** los pueda procesar. Resumiendo, el kernel utiliza sysfs para exportar nodos de dispositivos al espacio usuario y allí ser utilizados por **udev**.

Los device file son creados por el kernel a través del filesystem **devtmpfs**. Cualquier driver que quiera registrar un nodo de dispositivo debe pasar por devtmpfs para hacerlo. Cuando ya existe una instancia de *devtmpfs* montada en el directorio **/dev** el nodo dispositivo será creado inicialmente con un nombre, permisos y owner prefijados. Así, todos los nodos creados pertenecen a root y tienen el modo 0600 por defecto.

Acto seguido a la creación del nodo, el kernel envía un evento del tipo *uevent* a *udevd*. Basado en las reglas especificadas en los archivos dentro de */etc/udev/rules.d*, */lib/udev/rules.d* y */run/udev/rules.d*, *udevd* crea symlinks adicionales al nodo de dispositivo, o cambia sus permisos, dueño, grupo o modifica el nombre en la base de datos interna de *udev*. Si no existe ninguna regla, entonces el nodo queda tal cual como se creo.

## Creación del módulo class character

Para esta sección de práctica, se utilizará como base el código de la sección anterior. Pero a diferencia de crear el nodo de dispositivo manualmente, se creará por medio de devtmpfs.

En este caso el driver agrega una entrada al directorio **/sys/class**. Este directorio ofrece una vista de los drivers de dispositivos agrupados por clases.

Cuando la función register_chrdev_region() informa al kernel que hay un driver con un número major específico, no dice nada acerca del tipo de driver, por lo que no crea ninguna entrada en el directorio /sys/class. Esta entrada es necesaria para que mediante *devtmpfs* se cree un nodo de dispositivo dentro de */dev*.

Para la creación y destrucción de clases, el driver utiliza las siguientes funciones: **class_create()** y **class_destroy()**. Luego, para crear y destruir dispositivos se utilizan **device_create()** y **device_destroy()**.

Las diferencias entre la sección anterior y la actual son:

1. Se debe agregar el header correspondiente para el uso de las funciones mencionadas (*linux/device.h*).
2. El nombre de la clase y dispositivo deben estar definidos de antemano mediante un **#define** correspondiente.

3. Al no necesitar conocer los números major & minor, se puede hacer uso de la función **alloc_chrdev_region()**, para pasar esa información a **device_create()** y asignar ese dispositivo a la clase creada mediante **class_create()**.

Se invita al lector a descargar el archivo **holamundo_class_driver.c** y compilar el módulo mediante la técnica ya estudiada anteriormente. El programa **ioctl_test** se vuelve a utilizar en esta práctica, no es necesario recompilar.

Ejecute los siguientes comandos para cargar el módulo y verificar la creación de la clase en sysfs:

```
$ insmod holamundo_class_driver.ko

$ ls /sys/class
```

Una vez verificada la creación de la clase, se puede verificar el dispositivo creado, las entradas asociadas a ese dispositivo, y los números major y minor asociados.

```
$ ls /sys/class/hello_class

$ ls /sys/class/hello_class/mydev

$ cat /sys/class/hello_class/mydev/dev
```

También se puede verificar el nodo de dispositivo creado por *devtmpfs*:

```
$ ls -l /dev
```

Nuevamente, se corre el binario de pruebas **ioctl_test**, y luego de comprobar el funcionamiento, debe ejecutarse **rmmod** para limpiar la memoria.

# Creación de módulos que utilizan un framework

El framework **misc** es una interfaz exportada por el kernel de linux que permite que los módulos registren sus números minor individualmente.

Un device driver implementado como un miscellaneous character utiliza el número major ya asignado por el kernel para el tipo de dispositivos **misc**. Esto lo que hace a fin de cuentas es eliminar la necesidad de definir un número major único para el driver. Es muy importante desligar al desarrollador de esta responsabilidad dado que por el correr de los años y la gran cantidad de desarrolladores, las probabilidades de tener un conflicto de números major es cada vez mayor. La utilización de un dispositivo misc es una táctica efectiva para evitar los mencionados conflictos. A cada dispositivo que se prueba, se le asigna un número minor de manera dinámica, y se lista con una entrada de directorio dentro del filesystem virtual sysfs en la ruta */sys/class/misc*.

El major number **10** es asignado oficialmente para el driver misc. Cualquier módulo puede registrar números minor con el driver misc y tomar control sobre un dispositivo pequeño, que solamente necesite un entry point.

## Registrando un número minor con misc framework

Un dispositivo misc está definido por la estructura **miscdevice** definida en *include/linux/miscdevice.h*

```
struct miscdevice {
    int minor;
    const char *name;
    const struct file_operations *fops;
    struct list_head list;
    struct device *parent;
    struct device *this_device;
    const char *nodename;
    umode_t mode;
};
```

Donde:

- **minor** es el número minor que se está registrando.
- **name** es el nombre del dispositivo que luego será listado en el archivo **/proc/misc**.
- **fops** es un puntero a la estructura **file_operations** correspondiente.

- **parent** es un puntero a una estructura **device** que representa el dispositivo de hardware expuesto por este driver.

El driver misc exporta dos funciones para registrar el número minor que se desee y dar de baja el mismo: **misc_register()** y **misc_deregister()**. Existe la posibilidad de indicar a la función *misc_register()* que asigne un número minor de forma dinámica, indicando el número minor con el valor **MISC_DYNAMIC_MINOR**. La estructura que se pasa a la función *misc_register()* es linkeada en el kernel y no es destruida hasta tanto no se dé de baja.

Se invita al lector a descargar el código fuente **holamundo_misc_driver.c**, compilarlo como módulo, insertarlo en el kernel y verificar que se liste en el sysfs mediante los siguientes comandos:

```
$ insmod holamundo_misc_driver.ko
```

```
$ ls /sys/class/misc
```

Luego, verificar las entradas dentro del dispositivo creado y los números major y minor asociados:

```
$ ls /sys/class/misc/mydev
```

```
$ cat /sys/class/misc/mydev/dev
```

Es necesario verificar que también se haya creado el nodo en el devtmpfs antes de poder correr la aplicación de test:

```
$ ls -l /dev
```

Verificadas todas las entradas y los nodos, se debe proceder a ejecutar la aplicación **ioctl_test** y luego remover el módulo.