Taller de *drivers*

4 de octubre de 2016

Ejercicio

Se pide escribir el *driver* del dispositivo serial ficticio utilizando un *misc device*. Las aplicaciones deberán poder leer concurrentemente los datos que se reciben desde este dispositivo a través de un nodo `/dev/fakeserial`.

Ejemplo de uso:

```
$ sudo dd if=/dev/fakeserial bs=1 count=10
```

Tener en cuenta que:

- La acción de desencolar de la cola `kfifo` debe realizarse de forma exclusiva.
- No debe accederse a memoria de espacio de usuario directamente. El acceso debe ser a través de `copy_from_user()` o `copy_to_user()`.
- No debe hacerse *wait* de un semáforo al atender la interrupción.

Referencia resumida

Carga y descarga de módulos de Linux

Para la carga y descarga, se usan los comandos `insmod` y `rmmod`:

```
$ insmod fakeserial.ko
$ rmmod fakeserial
```

Ambos deben ser ejecutados como *root*.

Registro de un *misc device*

Para registrar y quitar un *misc device*, se usan las funciones:

```
int misc_register(struct miscdevice * misc);
int misc_deregister(struct miscdevice * misc);
```

La estructura `struct miscdevice` puede inicializarse del siguiente modo:

```
struct miscdevice mi_miscdev = {
    MI_MINOR,
    "midriver",
    &mi_fops,
};
```

El *minor* puede ser inicializado con `MISC_DYNAMIC_MINOR`, en cuyo caso el número será asignado por el kernel convenientemente. El tercer campo debe ser un puntero a un `struct file_operations`:

```
struct file_operations {
    struct module *owner;
    ...
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ...
}
```

Leer y escribir en memoria de usuario

Para leer y escribir en memoria de usuario, deben utilizarse siempre:

```
unsigned long copy_to_user(void __user * to, const void * from, unsigned long n);
unsigned long copy_from_user(void *to, const void __user * from, unsigned long n);
```

Dispositivo serial ficticio

El dispositivo serial debe ser inicializado con:

```
int fake_serial_init(int mode, fake_irq_handler_t irq_handler);
```

El único modo soportado actualmente es `FAKE_SERIAL_MODE_TEXT`.
El dispositivo genera una interrupcióncada vez que hay un byte listo para ser leido y llama a la función pasada en el parametro irq_handler. Esta deberá encargarsede leer el byte para que posteriormente sea leído.

Para obtener un byte desde el dispositivo serial, debe usarse:

```
u8 fake_inb(int port);
```

Esta función lee un byte del puerto de entrada-salida especificado; el correspondiente al dispositivo serial ficticio es: `FAKE_SERIAL_PORT`.

Antes de que el módulo sea descargado, debe llamarse a la función:

```
int fake_serial_exit(void);
```

Imprimir

Para hacer *debugging*, puede ser útil la función `printk()`. Su uso es similar al de `printf()` de la biblioteca de C, sólo que permite el agregado de un *flag* de prioridad.

```
printk(KERN_DEBUG "mimodulo: Error en la inicializacion\n");
```

La cadena formateada es copiada al *log buffer* del kernel, que normalmente es leído por una aplicación llamada `syslog`. Mirar los logs en `/var/logs/` o usar `dmesg` para ver los mensajes.

Colas en Linux

Las colas en Linux están representadas por la estructura `struct kfifo`. Para inicializar una cola, debe usarse:

```
int kfifo_alloc(struct kfifo *fifo, unsigned int size, gfp_t gfp_mask);
```

Esta función inicializa una cola de `size` bytes; para este ejercicio, una cola del tamaño de una página debería ser más que suficiente (`PAGE_SIZE`). `gfp_mask` indica flags para el mecanismo de reserva de memoria; con usar `GFP_KERNEL` debería alcanzar para el taller.

Para poder encolar y desencolar, pueden usarse las funciones:

```
unsigned int kfifo_in(struct kfifo *fifo, const void *from, unsigned int len);
unsigned int kfifo_out(struct kfifo *fifo, void *to, unsigned int len);
```

Tener en cuenta que, para el caso de un escritor y múltiples lectores, es necesario arbitrar la realización exclusiva de la lectura; no así de la escritura.

Semáforos en Linux

Los semáforos son representados mediante la estructura `struct semaphore`. Para inicializar un semáforo, puede usar:

```
void sema_init(struct semaphore *sem, unsigned int val);
```

La forma de hacer *wait* recomendada es mediante:

```
int down_interruptible(struct semaphore *sem);
```

El *signal* se realiza con:

```
void up(struct semaphore *sem);
```

Importante: Tener en cuenta que no se debe hacer *wait* sobre un semáforo al atender una interrupción. Esta operación sólo debe ser realizada en el contexto de un proceso, por ejemplo, en la operación de lectura del *misc device*.