Escucho y olvido, veo y recuerdo,
hago y entiendo. *

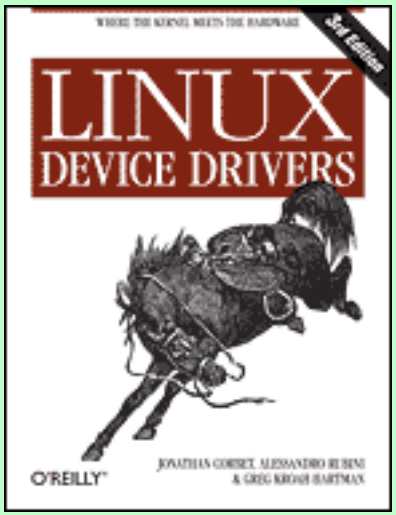
75-08 Sistemas Operativos
Lic. Ing. Osvaldo Clúa

2009

Facultad de Ingeniería
Universidad de Buenos Aires

Laboratorio: Construyendo un Driver para Linux 2.6

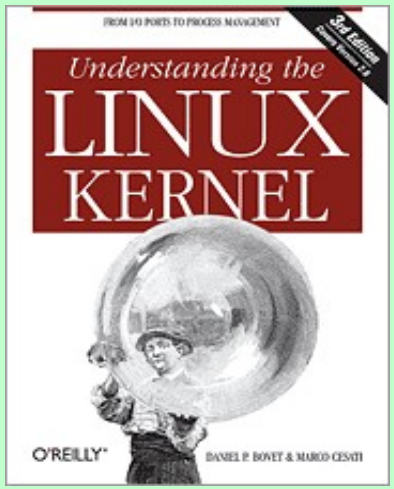
Lectura Obligatoria
(para quien quiera construir drivers)



Linux Device Drivers, Third Edition

Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman

Hay copia gratuita en la Web



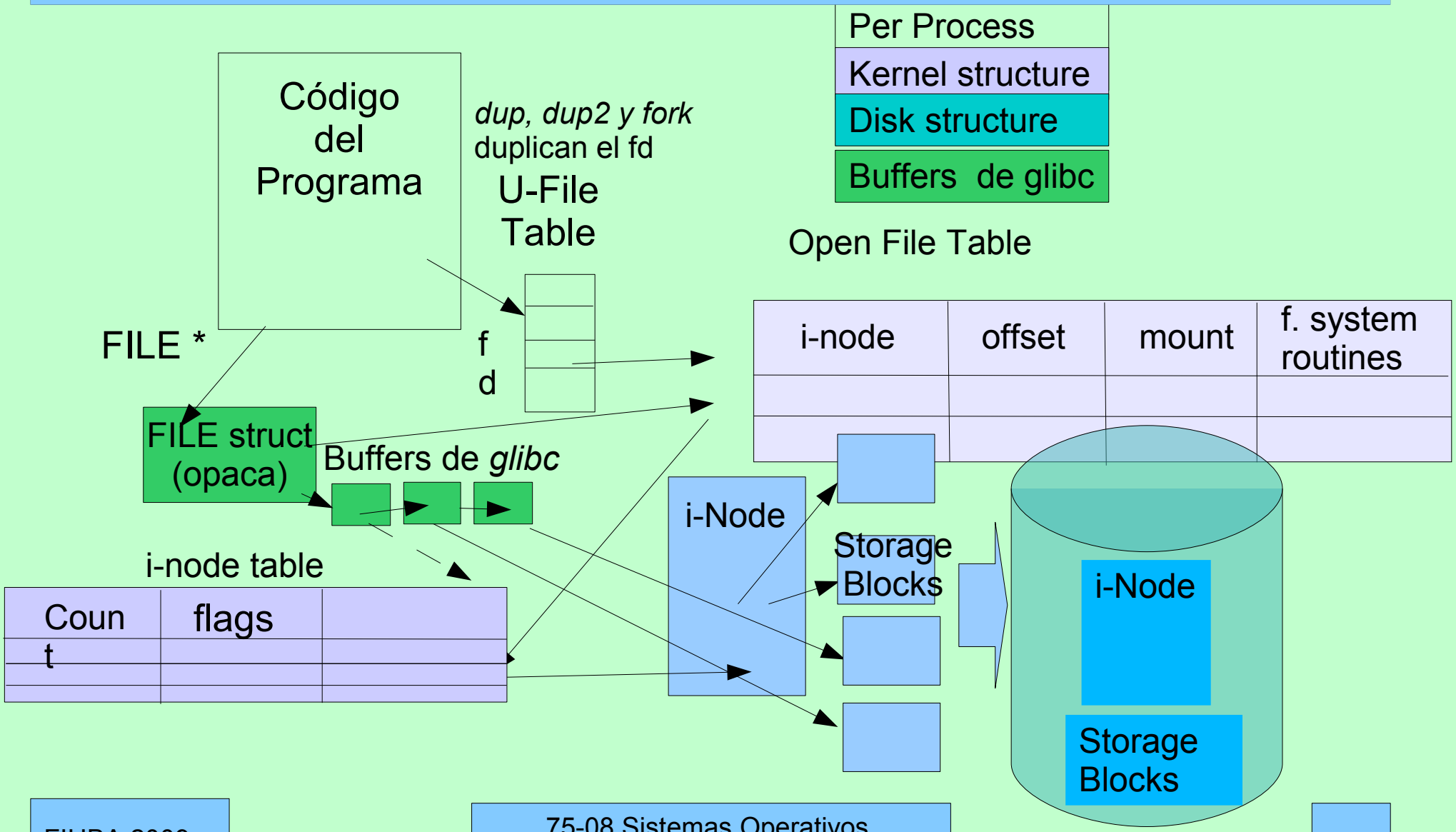
Understanding the Linux Kernel, Third Edition
By Daniel P. Bovet, Marco Cesati

El ejemplo desarrollado esta extraído/adaptado de estos libros

Introducción

- El Virtual File System depende de operaciones de bajo nivel para llevar a cabo sus operaciones (read, write, open, close, etc...).
- Estas operaciones provocan distintos cambios en las estructuras del Sistema Operativo y en las estructuras que implementan al File System.

Estructuras usadas en el procesamiento de Archivos



Device Driver Model

- Es el framework para el desarrollo de drivers.
  - Estructuras de Datos.
  - Funciones auxiliares.
- Ofrece una visión unificada de dispositivos, buses y drivers.
  - Y de sus interrelaciones.

sysfs

- Ofrece una visión del árbol del device driver model exportándolo al espacio de usuario.
  - Un directorio por cada objeto del árbol del modelo.
  - Links estableciendo las diversas relaciones (clases, buses, alarmas, subsistemas).
  - Archivos con los atributos de cada driver.

kobjects

- Encapsulan los datos relevantes a distintas partes del Kernel.
  - Los directorios del *sysfs* son *kobjects*.
  - Deben implementar una serie de rutinas y pueden estar compuestos solo por un conjunto determinado de tipos de datos.

ksets y subsistemas

- Son containers de kbjects y ksets respectivamente.
  - Los ksets agrupan kobjects del mismo tipo.
  - Un subsystem es una división mayor del kernel de Linux.
  - Todas estas estructuras tienen definidas rutinas para su manejo y administración.

El resto del driver-model

- El modelo se completa con
  - Devices – representan las capacidades de cada dispositivo
  - Bus – de acuerdo a la arquitectura (ATA, PCI)
  - Class – de acuerdo a la funcion (audio, red ...)
  - Intrerfaces – para comunicación con el ser humano (mouse, touch screen)
  - Driver – con los métodos para hot-plugging.

Device Driver

- Conjunto de rutinas que implementan los métodos de Virtual File System para controlar un dispositivo.
  - open(), close(), read(), ioctl(), lseek(), etc ...
  - Dependen de la *class* del dispsitivo y de su estrategia de uso (file system).
  - El device driver provee solo el mecanismo, las estrategias son provistas por otras partes del kernel

Loadable Kernel Modules

- Son object files que contienen código para extender al kernel (base-kernel) del Sistema Operativo.
  - Se carga en direcciones de memoria no contiguas (el kernel está cargado en direcciones contiguas)
  - Después de cargarse corren con el SisOp en modo Kernel y forman parte del kernel.

Linux Kernel API

- Son las funciones auxiliares que pueden usarse al programar Loadable Kernel Modules.
  - Mucho de la programación se basa en el uso de estas funciones.
  - Resuelven en forma unificada y segura el manejo de estructuras de datos.

Programación de un Loadable Kernel Module

- Se debe tener acceso a los fuentes del Kernel.
  - Ver la versión con `uname -r`
  - Se debe responder a dos eventos:
    - *load* del módulo - `insmod`
    - *unload* del módulo - `rmmod`

Módulo hola

hola.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

// para que no proteste por "Kernel contaminado"
MODULE_LICENSE("Dual BSD/GPL");

static int hola_comienzo(void) {
// No hay coma entre la prioridad del mensaje (<1>)
// y el msje
  printk("<1> Hola Che!\n");
  return 0;}

static void hola_final(void) {
  printk("<1> Adios, me las tomo\n");}

module_init (hola_comienzo);
module_exit (hola_final);
```

Makefile

```
KDIR:=/lib/modules/$(shell uname -r)/build

# hacer  mknod /dev/memory c 60 0

# obj-m implica loadable modules

obj-m:= hola.o
# -C indica que cambie a ese subdirectorio para
#  encontrar los  Makefiles a resolver antes de
# ejecutar el comando
# M= o SUBDIRS= le indica a kbuild que vaya al
# directorio antes de construir
# modules es el target por default

default:
	$(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
	$(RM) .*.cmd *.mod.c *.o *.ko -r .tmp*
```

Prueba de hola

La instalación debe hacerse como root

```
oclua@XPS-Desk:~/$ lsmod|grep hola
oclua@XPS-Desk:~/$ sudo insmod hola.ko
oclua@XPS-Desk:~/$ lsmod|grep hola
hola                    2944  0
oclua@XPS-Desk:~/$ sudo rmmod hola
```

En /var/log/kern.log

```
Jun 21 16:14:40 XPS-Desk kernel: [24704.418493]  Hola Che!
Jun 21 16:18:45 XPS-Desk kernel: [24949.197067]  Adios, me las tomo
```

Algunas notas sobre hola

- El makefile hace uso de las facilidades de `kbuild`.
  - Que cambia con la versión del Kernel.
- `module_init()` y `module_exit()` se llaman automáticamente al cargar y descargar el módulo.
  - Además de usar `lsmod`, se puede ver a hola en `/sys/modules`

Device Type

- Block Device
  - Pueden contener un file system.
- Character Device
  - Se ven como *streams*, de a un caracter por vez.

```
ls -l /dev/sc*
brw-rw----+ 1 root cdrom 11, 0 2009-06-21 06:22 /dev/scd0
brw-rw----+ 1 root cdrom 11, 1 2009-06-21 06:22 /dev/scd1
ls -l /dev/ttyS*
crw-rw---- 1 root dialout 4, 64 2009-06-21 06:22 /dev/ttyS0
crw-rw---- 1 root dialout 4, 65 2009-06-21 06:22 /dev/ttyS1
```

mknod

- Cada device tiene un par de números:
  - Major, que le indica al kernel a que drive pasar el comando
  - Minor que es tratado como parámetro para el driver.
- Estos archivos se hacen con `mknod major minor`

El Kernel llama al driver

- Ante distintos eventos
  - `open (fopen()), close (fclose()) , read (fread()), write (fwrite()) …`
- Y le pasa una `struct file_operations`
  - Que está en `include/linux/fs.h`
  - Hay una extension gnu y una C99 para poder inicializar este tipo de estructuras sin referirse a todos sus miembros.

Inicializacion de Tagged Structures en C

```
#include <stdio.h>

typedef struct  {
     int clase;      int roll;      char name[25];
}student;

void imp (student st){
       printf("( %d, %d, %s)",st.clase,st.roll,st.name);
       }
int main(){
       student st1={11,1,"Alan"};   // tradicional, vale en C y C++
       imp (st1); printf("\n");
       student st2={                          // roll no esta inicializado
              .name="Juan", .clase=4 };  // C99, no vale en C++
       imp (st2); printf("\n");
       student st3={
              name:"Pedro", clase:8};   // gnu C extension, no vale en C++
       imp (st3); printf("\n");
}
```

Character device que guarda un (1) byte en memoria

```
oclua@XPS-Desk:~/$ sudo mknod /dev/memoria c 60 0
oclua@XPS-Desk:~/$ ls -l /dev/memo*
crw-r--r-- 1 root root 60, 0 2009-06-21 18:36 /dev/memoria
oclua@XPS-Desk:~/$ sudo insmod memoria.ko

oclua@XPS-Desk:~/$ echo -n "Chau" >/dev/memoria   # solo guarda la “u”

oclua@XPS-Desk:~/$ cat /dev/memoria  # solo sale la u
uoclua@XPS-Desk:~/$
oclua@XPS-Desk:~/$ dd < /dev/memoria
u0+1 records in
0+1 records out
1 byte (1 B) copied, 1.5855e-05 s, 63.1 kB/s
oclua@XPS-Desk:~/$ sudo rmmod memoria
```

En /var/log/kern.log

```
Jun 21 19:17:12 XPS-Desk kernel: [35628.065001] Cargando memoria module
Jun 21 19:17:47 XPS-Desk kernel: [35662.945419] Descargando memoria module
```

memoria.c
init

```
/* Estructura de las
operaciones*/

struct file_operations
memoria_fops = {
  read: memoria_read,
  write: memoria_write,
  open: memoria_open,
  release: memoria_release
};

module_init(memoria_init);
module_exit(memoria_exit);

/* Major number */
int memoria_major = 60;
/* Buffer */

char *memoria_buffer;
```

```
int memoria_init(void) {
  int result;   /* Registrando device */
  result = register_chrdev(memoria_major, "memoria",
&memoria_fops);
  if (result < 0) {  printk(
      "<1>memoria: no se obtiene el major number %d\n",
memoria_major);
    return result;
  }
  /* Reservar memoria para el buffer */
  memoria_buffer = kmalloc(1, GFP_KERNEL);
  if (!memoria_buffer) {
    result = -ENOMEM;
    memoria_exit();
    return result;
  }
  memset(memoria_buffer, 0, 1);
  printk("<1>Cargando memoria module\n");
  return 0;
}
```

memoria.c
exit, open y release

```
void memoria_exit(void) {
  /* Liberar el major number */
  unregister_chrdev(memoria_major, "memoria");
  /* Liberar el buffer */
  if (memoria_buffer) {
    kfree(memoria_buffer);
  }
  printk("<1>Descargando memoria module\n");
}

int memoria_release(struct inode *inode, struct file *filp) {
  return 0;
}

int memoria_open(struct inode *inode, struct file *filp) {
  return 0;
}
```

memoria.c
read y write

```
ssize_t memoria_read(struct file *filp, char *buf,  size_t count, loff_t *f_pos)
 {
                                        /* Pasar datos al user space */
  copy_to_user(buf,memoria_buffer,1);
                                       /* Cambiar posicion de lectura */
  if (*f_pos == 0) {
       *f_pos+=1;
    return 1;    // un caracter leido
  } else {
    return 0;   // eof (vacio)
  }
}

ssize_t memoria_write( struct file *filp, char *buf, size_t count, loff_t *f_pos) {
  char *tmp;
  tmp=buf;                                                  // el primer caracter
  copy_from_user(memoria_buffer,tmp,1);
  return 1;
}
```

Algo de debug

- El write escribe en memoria el primer caracter.
  - Pero el read recupera el último.

```
oclua@XPS-Desk:~$strace echo -n "Chau" >/dev/memoria
------------------------------------------------------------------------------
write(1, "Chau", 4)                      = 1
write(1, "hau", 3)                       = 1
write(1, "au", 2)                        = 1
write(1, "u", 1)                         = 1
close(1)                                 = 0
------------------------------------------------------------------------------
```

Uso de /dev/memoria por programas

escritura.cc

```
....
int main(int argct, char * argv[]){
        if (argct !=3) {
        .....  }
        ofstream sal(argv[1]);
        string s(argv[2]);
        sal<<s;
        cout<<"Escrito "<<s<<" en "<<argv[1]<<endl;
}
```

lectura.cc

```
int main(int argct, char * argv[]){
        if (argct !=2) {....
                }
        ifstream ent (argv[1]);
        string s;
        ent>>s;
        cout<<"Leido "<<s<<" de "<<argv[1]<<endl;
}
```

```
oclua@XPS-Desk:~$ ./escritura /dev/memoria k
Escrito k en /dev/memoria
oclua@XPS-Desk:~$ ./lectura /dev/memoria
Leido k de /dev/memoria
```