



Devices en Kernel Linux

Devices en Linux Kernel 2.4.x



Devices en Kernel Linux

Tipos de Devices

- Devices de caracteres

- ◆ Terminales, Seriales

```
crw-rw---- 1 root dialout 4, 64 2004-03-10 07:34 ttyS0
```

- Devices de bloques

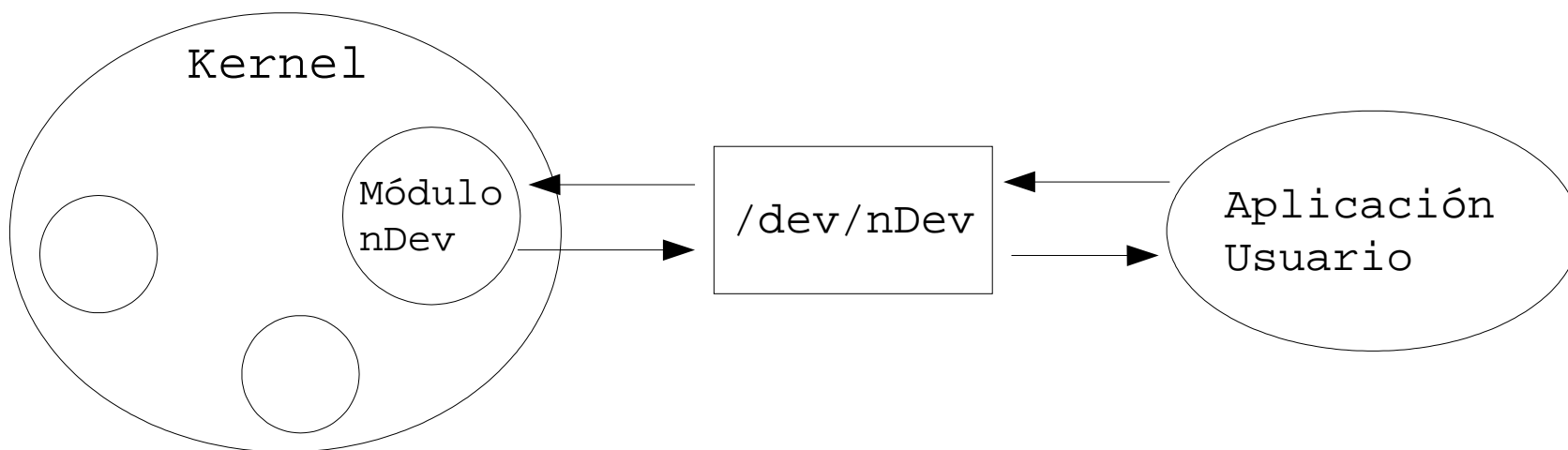
- ◆ Discos duros

```
brw-rw---- 1 root disk 3, 1 2004-03-10 07:33 hda1
```



Devices y Módulos

- Los dispositivos suelen ser controlados por módulos del kernel.





Operaciones Básicas

- Como el un `/dev` se accesa de la misma manera que un archivo, el modulo asociado debe soportar las operaciones:
 - ♦ Read, Write
 - ♦ Open, Close
- Además los modulos pueden ser insertados y eliminados del kernel. Por lo que debe soportar
 - ♦ Inicialización (Variables, memoria estructuras, etc...)
 - ♦ Salida (Limpieza de memoria, etc...)



Operaciones

- **Open**

- ◆ `int nDev_open (struct inode *inode, struct file *filp)`

- **Release**

- ◆ `int nDev_release (struct inode *inode, struct file *filp)`

- **Read**

- ◆ `ssize_t nDev_read (struct file *filp, char *buf, size_t count, loff_t *f_pos)`

- **Write**

- ◆ `ssize_t nDev_write (struct file *filp, const char *buf, size_t count, loff_t *f_pos)`



Devices en Kernel Linux

Operaciones

- **Init**
 - ◆ `int nDev_init(void);`
- **Exit**
 - ◆ `void nDev_cleanup(void);`



Operaciones

- Registro de funciones

```
module_init(nDev_init);  
module_exit(nDev_cleanup);
```

```
struct file_operations nDev_fops=  
{  
    read: nDev_read,  
    write: nDev_write,  
    open: nDev_open,  
    release: nDev_release,  
};
```



Devices en Kernel Linux

Variables Globales

```
int nDev_major = NDEV_MAJOR;
int num_devs = NUM_DEVS; /* Numero de devices nDev*/
int bufferSize = BUFFERSIZE;

MODULE_PARM(nDev_major, "i");
MODULE_PARM(num_devs, "i");
MODULE_PARM(bufferSize, "i");

MODULE_AUTHOR("Mario Leyton");
MODULE_LICENSE("GPL");

Nano_Dev *nano_devices;
```




Devices en Kernel Linux

Init

```
int nDev_init(void){
    int result, i;

    SET_MODULE_OWNER(&nDev_fops);

    /*Registramos el numero Major y aceptamos un numero dinamico.*/
    result = register_chrdev(nDev_major, "nDev", &nDev_fops);
    if (result < 0) return result;
    if (nDev_major == 0) nDev_major = result; /* dinamico */

    /* Pedimos memoria para los devices con kmalloc */
    nano_devices=kmalloc(num_devs * sizeof(Nano_Dev), GFP_KERNEL);
    if (!nano_devices) {
        unregister_chrdev(nDev_major, "nDev");
        return -ENOMEM;
    }
    memset(nano_devices, 0, num_devs * sizeof (Nano_Dev));
    for(i=0; i < num_devs; i++){
        nano_devices[i].buff_use=0;
        sema_init (&nano_devices[i].sem, 1);
    }
    return 0; /* ok */
}
```



Devices en Kernel Linux

Exit

```
void nDev_cleanup(void)
{
    unregister_chrdev(nDev_major, "nDev");
    kfree(nano_devices);
}
```



Devices en Kernel Linux

Open

```
int nDev_open (struct inode *inode, struct file *filp)
{
    int num = MINOR(inode->i_rdev);
    Nano_Dev *dev; /* Puntero a la estructura del dispositivo*/

    /* Vemos cual dispositivo se esta abriendo*/
    if (num >= num_devs) return -ENODEV;
    dev = &nano_devices[num];

    /* Guardamos un puntero al nDev en filp->private_data
     * para ser recuperado en read/write */
    filp->private_data = dev;

    MOD_INC_USE_COUNT;
    return 0;          /* exitoso */
}
```



Devices en Kernel Linux

Read

```
ssize_t nDev_read (struct file *filp, char *buf, size_t count,
                  loff_t *f_pos){
    /* recuperamos la estructura del nDev */
    Nano_Dev *dev = filp->private_data;
    if (down_interruptible (&dev->sem)) return -ERESTARTSYS;

    /* vemos los casos patologicos */
    if( *f_pos >= dev->buff_use){
        up (&dev->sem);
        return 0;
    }
    if( *f_pos+count > dev->buff_use) count = dev->buff_use - *f_pos;
    if (copy_to_user (buf, &dev->buffer[*f_pos], count)) {
        up(&dev->sem);
        return -EFAULT;
    }
    up (&dev->sem);
    *f_pos += count;
    return count;
}
```



Devices en Kernel Linux

Write

```
ssize_t nDev_write (struct file *filp, const char *buf, size_t count,
                    loff_t *f_pos)
{
    Nano_Dev *dev = filp->private_data;
    if (down_interruptible (&dev->sem)) return -ERESTARTSYS;

    if(*f_pos==0) dev->buff_use=0; /* Primera escritura */

    if(dev->buff_use > bufferSize || dev->buff_use+count > bufferSize){
        up (&dev->sem);
        return -ENOMEM;
    }
    if (copy_from_user (dev->buffer + *f_pos, buf, count)) {
        up (&dev->sem);
        return -EFAULT;
    }

    *f_pos += count;
    dev->buff_use += count;
    dev->buffer[dev->buff_use]='\0';
    up (&dev->sem);
    return count;
}
```



Devices en Kernel Linux

Close

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



Devices en Kernel Linux

Uso

- Requisitos
 - ♦ Kernel headers en `/usr/src/linux`
- Make
 - ♦ `make dep`
 - ♦ `make`
- Load (*como root*)
 - ♦ `./nDev_load`
 - ♦ `./nDev_unload`
- Test
 - ♦ `make test`
 - ♦ `./test`