Sistemas Operativos

Kernel II – Módulos y drivers

Facultad de Informática
UNIVERSIDAD NACIONAL DE LA PLATA

Sistemas Operativos

Módulos y Drivers

Módulos

- "Pedazos de código" que pueden ser cargados y descargados bajo demanda.
- Extienden la funcionalidad del kernel.
- Sin ellos el kernel sería 100% monolítico
- Todo el soporte debería estar en la imagen del kernel.
- No recompilar.
- No rebootear.

Módulos

- Comandos relacionados a módulos
  - lsmod

  Lista los módulos cargados (es equivalente a cat /proc/modules).
  - rmmod módulos

  Descarga uno o más módulos.
  - modinfo modulo

  Nos muestra información sobre el módulo
  - insmod módulo [opciones]

  Trata de cargar el módulo especificado.
  - depmod
  - Hay dependencias que deben respetarse al cargar y descargar módulos. depmod permite calcular tales dependencias. Por defecto depmod -a escribe las dependencias en el archivo /lib/modules/version/modules.emp
  - modprobe módulo opciones

  Emplea la información de dependencias generada por depmod e información de /etc/modules.conf para cargar el módulo especificado.

¿Como escribir un módulo?

- Debemos proveer dos funciones:
- Inicialización: Ejecutada cuando ejecutamos insmod.
- Descarga: Ejecutada cuando ejecutamos rmmod.

```
/* Se necesita siempre modules */
#include <linux/module.h>
#include <linux/kernel.h>
Int init_module(void){

printk(KERN_INFO "Hello world 1.\n");
return 0;}

void cleanup_module(void){
printk(KERN_INFO "Goodbye world 1.\n");}
```

¿Como escribir un módulo?

- También podemos indicarle otras funciones.

```
#include <linux/module.h> /* Se necesita siempre */
#include <linux/kernel.h> /* Para KERN_INFO */
#include <linux/init.h> /* Para las macros */

static int __init hello_2_init(void){
printk(KERN_INFO "Hello, world 2\n");
return 0;}

static void __exit hello_2_exit(void){
printk(KERN_INFO "Goodbye, world 2\n");}

module_init(hello_2_init);
module_exit(hello_2_exit);
```

¿Como escribir un módulo?

Construir el Makefile

```
obj-m += hello-1.o
all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
modules
clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Compilar: make

```
make -C /lib/modules/2.6.28.1/build M=/root/lkmpg-examples/02-HelloWorld modules
make[1]: Entering directory `/usr/src/linux-2.6.28.1'
CC [M] /root/lkmpg-examples/02-HelloWorld/hello-1.o
Building modules, stage 2.
MODPOST
CC /root/lkmpg-examples/02-HelloWorld/hello-1.mod.o
LD [M] /root/lkmpg-examples/02-HelloWorld/hello-1.ko
make[1]: Leaving directory `/usr/src/linux-2.6.28.1'
hostname:~/lkmpg-examples/02-HelloWorld#
```

Dispositivos

- Entendemos por dispositivo a cualquier dispositivo de hard: discos, memoria, mouse, etc.
- Cada operación sobre un dispositivo es llevada por código específico para el dispositivo.
- Este código se denomina “driver”.

Dispositivos en UNIX

- Cada dispositivo de hardware es un archivo (abstracción).
- Ejemplo: /dev/hda.
  - En realidad no es un archivo.
  - Si leemos/escribimos desde él lo hacemos sobre datos “crudos” del disco (bulk data).
- Accedemos a estos archivos mediante operaciones básicas (espacio del kernel).
  - read, write: escribir y recuperar bulk data
  - ioctl: configurar el dispositivo

Dispositivos en UNIX

- Podemos clasificar el hard en varios tipos.
  - Dispositivos de acceso aleatorio(ej. discos). Dispositivos seriales(ej. Mouse, sonido,etc).
- Acorde a esto clasificamos los drivers.
  - Dispositivos de bloques: son un grupo de bloques de datos persistentes. Leemos y escribimos de a bloques, generalmente de 1024 bytes.
  - Dispositivos de caracter: Se accede de a 1 byte a la vez y 1 byte sólo puede ser leído por única vez.

Dispositivos en UNIX

- Major y Minor device number.
  - Los dispositivos se dividen en números llamados major device number. Ej: los discos SCSI tienen el major number 8.
  - Cada dispositivo tiene su minor device number.
  - Ejemplo: /dev/sda major number 8 y minor number 0.
- Con el major y el minor number el kernel identifica un dispositivo.
- <kernel_code>/linux/Documentation/devices.txt.

Dispositivos en UNIX

- Major y Minor device number.
  - Los dispositivos se dividen en números llamados major device number. Ej: los discos SCSI tienen el major number 8.
  - Cada dispositivo tiene su minor device number.
  - Ejemplo: /dev/sda major number 8 y minor number 0.
- Con el major y el minor number el kernel identifica un dispositivo.
- <kernel_code>/linux/Documentation/devices.txt.

Dispositivos en UNIX

```
# ls -l /dev/hda[1-3]
brw-rw---- 1 root disk 3, 1 Abr 19 15:24 /dev/hda1
brw-rw---- 1 root disk 3, 2 Abr 19 15:24 /dev/hda2
brw-rw---- 1 root disk 3, 3 Abr 19 15:24 /dev/hda3
```

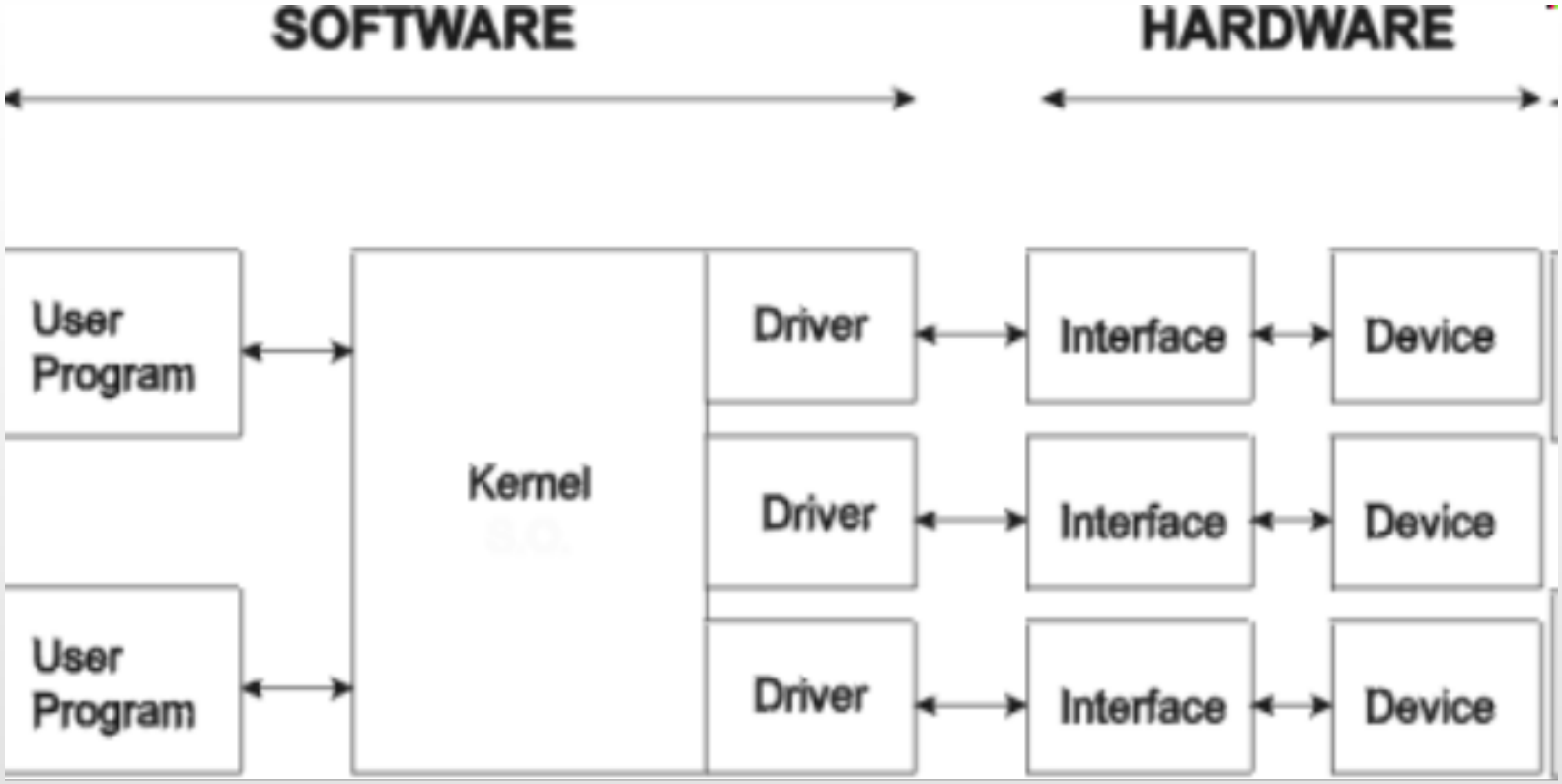
Device Files

- Representación de los dispositivos
- Por convención están en el /dev
- Se crean mediante el comando mknod.

```
mknod [-m <mode>] <file-name> [b|c] <major-number> <minor-number>
```

b o c: según se trate de dispositivos de caracter o de bloque.
El minor y el major number lo obtenemos de <kernel_code>/linux/Documentation/devices.txt

Dispositivos en UNIX



Dispositivos en UNIX

- Necesitamos decirle al kernel:
  - Que hacer cuando se escribe al device file.
  - Que hacer cuando se lee desde el device file.
- Esto lo hacemos en un módulo.
- Los drivers se implementan utilizando módulos de carga dinámica.

- La struct file_operations:
  - Sirve para decirle al kernel como leer/ escribir al dispositivo.
  - Cada variable posee un puntero a las funciones que implementan las operaciones sobre el dispositivo.
- Dispositivos de caracter
  - Se lee un flujo de bytes.
  - Se lee de a 1 caracter único por vez.

Dispositivos en UNIX

- Mediante la struct file_operations especifico que funciones leen/ escriben al dispositivo.

```
struct file_operations my_driver_fops = {
read: myDriver_read,
write: myDriver_write,
open: myDriver_open,
release: mydriver_release};
```

- En el module_init registro mi driver

```
register_chrdev(major_number, "myDriver", & my_driver_fops);
```

- En el module_exit registro mi driver

```
unregister_chrdev(major_number, "myDriver");
```

- Operaciones sobre el dispositivo
  - Escritura del archivo de dispositivo (Ej. echo "hi" > /dev/myDeviceFile)

```
ssize_t myDriver_read(struct file *filp, char *buf,size_t count, loff_t *f_pos)
```

- Lectura del archivo de dispositivo (cat /dev/myDeviceFile)

```
ssize_t myDriver_read(struct file *filp, char *buf,size_t count, loff_t *f_pos)
```

Parámetros de las funciones

- Struct file: Estructura del kernel que representa un archivo abierto.
- char *buf: El dato a leído o a escribir desde/hacia el dispositivo(espacio de usuario) .
- size_t count: La longitud de buf.
- loff_t *f_pos count: La posición manipulada

ssize_t dev_read(struct file *file, char *buf, size_t count, loff_t *ppos);