Sistemas Operativos 2015

Trabajo Práctico Nro 3: Desarrollo de módulos para el kernel de Linux

Resumen: *El objetivo de este trabajo práctico es el desarrollo de dos simples módulos para insertar en el kernel de linux. Estos módulos van a emular el funcionamiento de dos dispositivos de caracteres. Uno de los dispositivos realizará una encriptación simple de los caracteres que se escriben en el. El otro de los módulos realizará la desencriptación de los caracteres que se escriben en el.*

Introducción:

Los sistemas operativos basados en el kernel Linux funcionan correctamente en una gran variedad de plataformas de hardware, diferentes placas de video, diferentes placas de red, diferentes discos duros, etc. La mayor parte de las aplicaciones de estos sistemas operativos son desarrolladas sin conocer las características del hardware sobre el que serán utilizadas. Es la tarea del kernel Linux la de actuar de interfáz entre las aplicaciones y las diferentes plataformas de hardware. Gran parte de la tarea de comunicación entre el kernel y los diferentes dispositivos físicos es realizada por los controladores de dispositivos o módulos.

La tarea de los módulos es presentar al kernel un conjunto estandarizado y bien definido de llamadas al sistema, estas llamadas son independientes del tipo de dispositivo. El módulo luego traduce estas llamadas a las operaciones específicas del hardware para el que fue diseñado.

Este conjunto de llamadas al sistema esta definida de tal manera que los módulos pueden ser desarrollados de manera separada del resto del kernel. Estos módulos son cargados en tiempo de ejecución en el momento en que son necesarios.

Como se carga un módulo?

El proceso de carga típico de un módulo (simplificado) es el siguiente. Cuando el kernel necesita una funcionalidad que no esta presente, el demonio de módulos del kernel *kmod* ejectua el binario modprobe para cargar el módulo necesario.

Claramente *kmod* “sabe” cual es el módulo que necesita cargar para satisfacer una necesidad particular. El programa modprobe verifica si el módulo en cuestion necesita

de la presencia de otros módulos para funcionar correctamente. Una vez que ha definido cuales son los módulos que deben ser cargados y donde se encuentran ubicados modprobe utiliza el programa insmod para cargar estos módulos en memoria y ponerlos a disposicion del kernel para ser utilizados. Podemos ver que módulos estan cargados en este momento y cuales son sus dependencias con el comando *lsmod*

Como se compila un módulo?

Como indicamos arriba los módulos utilizan para comunicarse con el kernel una interfaz previamente definida, esta interfaz esta descripta en los archivos de header del kernel mismo. Para compilar nuestro propio módulo vamos a necesitar entonces los headers del kernel que estamos corriendo.

Primero averiguemos que kernel estamos corriendo:

```
hugo@marvin:~$ uname -r
2.6.32-5-686
hugo@marvin:~$
```

Ahora instalemos los headers de este kernel. En el caso de sistemas basados en Debian podemos usar aptitude para averiguar que paquete tenemos que instalar, por ejemplo:

```
hugo@marvin:~$ aptitude search 2.6.32-5-686 |grep header
i   linux-headers-2.6.32-5-686      - Header files for Linux 2.6.32-5-686
p   linux-headers-2.6.32-5-686-bigm - Header files for Linux 2.6.32-5-686-bigmem
hugo@marvin:~$
```

Instalamos el paquete con ***aptitude install*** *linux-headers-2.6.32-5-686*

Ya tenemos las herramientas necesarias para compilar nuestro módulo. Escribamos ese módulo de ejemplo para probar, este ejemplo esta tomado literalmente de (http://tldp.org/LDP/lkmpg/2.6/html/x121.html)

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

int init_module(void)
{
        printk(KERN_INFO "Hola Mundo!\n");
        return 0;
}

void cleanup_module(void)
{
        printk(KERN_INFO "Adios Mundo Cruel!\n");
}
```

De estas pocas lineas podemos aprender mucho, vemos que no hay una funcion main(). Solo estan dos funciones init_module() y cleanup_module() estas dos funciones tienen que estar definidas si o si y son las que se ejecutan al cargar el módulo y al descargarlo. Tambien vemos la funcion printk() esta es una funcion que el kernel utiliza para realizar el registro de eventos es por eso que junto con el mensaje se debe dar ademas una prioridad. (ver `linux/kernel.h` para mas detalles).

Guardemos este codigo en un archivo por ejemplo `hola.c` y veamos como compilarlo. Debajo tenemos un ejemplo de Makefile para compilar este pequeño módulo que hemos escrito

obj-m += hola.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
```

Tratemos de compilar el módulo y veamos que pasa

```
hugo@marvin:/tmp/aa$ make
make -C /lib/modules/2.6.32-5-686/build M=/tmp/aa modules
```

```
make[1]: Entering directory `/usr/src/linux-headers-2.6.32-5-686'
  CC [M]  /tmp/aa/hola.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /tmp/aa/hola.mod.o
  LD [M]  /tmp/aa/hola.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.32-5-686'
hugo@marvin:/tmp/aa$
```

Ahora solo resta cargar el módulo y verlo funcionar. Dado que el codigo que ejecutan los módulos tiene acceso al espacio de memoria del kernel no cualquier usuario puede cargar un módulo en memoria, para cargar nuestro módulo debemos hacerlo como el usuario root.

```
root@marvin:/tmp/aa# insmod hola.ko
```

Ahora veamos si la carga funciono:

```
root@marvin:/tmp/aa# lsmod |head
Module                  Size  Used by
hola                    532  0
decript                 1997  0
encript                 1997  0
nls_utf8                 908  0
nls_cp437               4489  0
vfat                    6534  0
fat                    34776  1 vfat
uvcvideo               45526  0
sky2                   33904  0
root@marvin:/tmp/aa#
```

Y veamos nuestro mensaje en el archivo de mensajes del kernel

```
root@marvin:/tmp/aa# tail -n 1 /var/log/messages
Sep  9 16:56:27 marvin kernel: [71918.042017] Hola Mundo!
```

Y eso es todo, si ahora descargamos el módulo veremos el mensaje de despedida

```
root@marvin:/tmp/aa# rmmod hola
root@marvin:/tmp/aa# tail -n 1 /var/log/messages
Sep  9 16:57:39 marvin kernel: [71990.230281] Adios Mundo Cruel!
root@marvin:/tmp/aa#
```

Manejadores de dispositivos de tipo caracter

Las operaciones que se pueden realizar sobre un dispositivo de tipo caracter estan definidas por el kernel asi como el nombre de las funciones que implementan estas operaciones ver *linux/fs.h.*

Por ejemplo un manejador tiene que definir una funcion que lea del dispositivo esa funcion debe llamarse *read* segun esta definido en *fs.h*

No todos los manejadores deben implementar todas las funciones, por ejemplo un manejador de una placa de video no necesita implementar una funcion para leer una estructura de directorios (*readdir*)

Registrando un dispositivo

Los dispositivos de caracteres son utilizados accediendo a archivos de dispositivo. Estos archivos estan ubicados en /dev

```
hugo@marvin:~/docencia/SO1/2011/practicos/módulo$ ls -l /dev
total 0
crw-rw----+ 1 root audio    14,   4 Oct  6 14:24 audio
crw-------  1 root root      5,   1 Oct  6 14:25 console
crw-------  1 root root     10,  62 Oct  6 11:24 cpu_dma_latency
crw-rw----+ 1 root audio    14,   3 Oct  6 14:24 dsp
```

Agregar un manejador al sistema significa registrarlo con el kernel, al realizar esta operacion el kernel asigna un nro (Major number) que identifica al manejador, ver /proc/devices

Para registrar el dispositivo se utiliza la funcion `register_chrdev` definida en fs.h

Tareas:

- Escribir un manejador de dispositivos de caracter que cuando se escriba una cadena de caracteres en su archivo de dispositivo tome esa cadena y la “encripte” sumando un entero fijo a cada caracter de la cadena.
- Cuando se lea del archivo de dispositivo encriptador el manejador entrege la ultima cadena encriptada.
- Escribir un manejador de dispositivos de caracter que cuando se escriba una cadena de caracteres en su archivo de dispositivo tome esa cadena y la “desecnripte” restando el mismo entero del primer manejador a cada caracter de la cadena.
- Cuando se lea del archivo de dispositivo desencriptador el manejador entrege la ultima cadena desencriptada.

NOTA:

Todo lo necesario para resolver este practico se puede encontrar en:
http://tldp.org/LDP/lkmpg/2.6/html/index.html