Sistemas Operativos 2019

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 interfaz 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 está 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 está presente, el demonio de módulos del kernel kmod ejecuta 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 cuestión 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 disposición del kernel para ser utilizados. Podemos ver que módulos están 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 está descrita 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:

```
malonso@qui-gon ~$ uname -r
5.3.1-arch1-1-ARCH
malonso@qui-gon ~$
```

Ahora instalemos los headers de este kernel. En el caso de sistemas basados en Arch podemos usar *pacman -Ss linux-headers* para averiguar que paquete tenemos que instalar, por ejemplo:

```
malonso@qui-gon ~$ sudo pacman -Ss linux-headers
core/linux-headers 5.3.1.arch1-1 [installed]
      Header files and scripts for building modules for Linux kernel
malonso@qui-gon ~$
```

En sistemas basados en debian podemos usar *apt search linux-headers.*
Instalamos el paquete con *pacman -S* en sistemas basados en Arch o aptitude install en sistemas basados en Debian, o con el manejador de paquetes correspondiente a la distro.

Ya tenemos las herramientas necesarias para compilar nuestro módulo. Escribamos ese módulo de ejemplo para probar, este ejemplo esta tomado literalmente de (https://static.lwn.net/images/pdf/LDD3/ch02.pdf)

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

```
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
   printk(KERN_ALERT "Hello, world\n");
   return 0;
}

static void hello_exit(void)
{
   printk(KERN_ALERT "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

De estas pocas líneas podemos aprender mucho, vemos que no hay una función main(). Solo están dos funciones *hello_init()* y *hello_exit()* estas dos funciones se registran con *module_init()* y *module_exit()* respectivamente y son las que se ejecutan al cargar el módulo y al descargarlo. También vemos la función *printk()* esta es una función que el kernel utiliza para realizar el registro de eventos es por eso que junto con el mensaje se debe dar además una prioridad. (ver linux/kernel.h para mas detalles).
Guardemos este código en un archivo por ejemplo *hello.c* y veamos cómo compilarlo. Debajo tenemos un ejemplo de Makefile para compilar este pequeño módulo que hemos escrito

```
#if KERNELRELEASE is defined, we've been invoked from the
# kernel build system and can use its language.

ifneq ($(KERNELRELEASE),)
      obj-m := hello.o


# Otherwise we were called directly from the command
# line; invoke the kernel build system.
else
      KERNELDIR ?= /lib/modules/$(shell uname -r)/build
      PWD := $(shell pwd)

default:
      $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif
```

Tratemos de compilar el módulo y veamos que pasa

```
malonso@qui-gon tp3$ make
make -C /lib/modules/5.3.1-arch1-1-ARCH/build
M=/home/malonso/Documents/SO1/2019/tp3 modules
make[1]: Entering directory '/usr/lib/modules/5.3.1-arch1-1-ARCH/build'
  CC [M]  /home/malonso/Documents/SO1/2019/tp3/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/malonso/Documents/SO1/2019/tp3/hello.ko
make[1]: Leaving directory '/usr/lib/modules/5.3.1-arch1-1-ARCH/build'
malonso@qui-gon tp3$
```

Notemos que ahora tenemos nuestro módulo *hello.ko*

```
malonso@qui-gon tp3$ ls -l
total 36
-rw-r--r-- 1 malonso users  313 Oct 24 16:05 hello.c
-rw-r--r-- 1 malonso users 4232 Oct 24 16:05 hello.ko
-rw-r--r-- 1 malonso users   46 Oct 24 16:05 hello.mod
-rw-r--r-- 1 malonso users  646 Oct 24 16:05 hello.mod.c
-rw-r--r-- 1 malonso users 2776 Oct 17 17:33 hello.mod.o
-rw-r--r-- 1 malonso users 2328 Oct 24 16:05 hello.o
-rw-r--r-- 1 malonso users  406 Oct 17 17:32 Makefile
-rw-r--r-- 1 malonso users   46 Oct 24 16:05 modules.order
-rw-r--r-- 1 malonso users    0 Oct 17 17:33 Module.symvers
malonso@qui-gon tp3$
```

Ahora solo resta cargar el módulo y verlo funcionar. Dado que el código 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@qui-gon tp3]# insmod ./hello.ko
[root@qui-gon tp3]#
```

Ahora veamos si la carga funciono:

```
[root@qui-gon tp3]# lsmod | head
Module                  Size  Used by
hello                  16384  0
```

```
snd_usb_audio           278528  1
snd_usbmidi_lib         40960  1 snd_usb_audio
snd_rawmidi             45056  1 snd_usbmidi_lib
snd_seq_device          16384  1 snd_rawmidi
uinput                  20480  1
rfcomm                  90112  7
fuse                    139264  3
8021q                   40960  0
[root@qui-gon tp3]#
```

Y veamos nuestro mensaje en el archivo de mensajes del kernel

```
[root@qui-gon tp3]# dmesg | tail
[43137.395111] audit: type=1101 audit(1571942464.486:182): pid=17553
uid=1000 auid=4294967295 ses=4294967295 msg='op=PAM:accounting acct="root"
exe="/bin/su" hostname=? addr=? terminal=/dev/pts/0 res=success'
[43137.396001] audit: type=1103 audit(1571942464.490:183): pid=17553
uid=1000 auid=4294967295 ses=4294967295 msg='op=PAM:setcred acct="root"
exe="/bin/su" hostname=? addr=? terminal=/dev/pts/0 res=success'
[43137.397164] audit: type=1105 audit(1571942464.490:184): pid=17553
uid=1000 auid=4294967295 ses=4294967295 msg='op=PAM:session_open
acct="root" exe="/bin/su" hostname=? addr=? terminal=/dev/pts/0
res=success'
[44216.914544] audit: type=1106 audit(1571943544.019:185): pid=17553
uid=1000 auid=4294967295 ses=4294967295 msg='op=PAM:session_close
acct="root" exe="/bin/su" hostname=? addr=? terminal=/dev/pts/0
res=success'
[44216.916260] audit: type=1104 audit(1571943544.019:186): pid=17553
uid=1000 auid=4294967295 ses=4294967295 msg='op=PAM:setcred acct="root"
exe="/bin/su" hostname=? addr=? terminal=/dev/pts/0 res=success'
[45227.055118] Hola Mundo!
```

Manejadores de dispositivos de tipo caracter

Las operaciones que se pueden realizar sobre un dispositivo de tipo caracter están 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 función 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 función para leer una estructura de directorios (readdir)

Registrando un dispositivo

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

```
malonso@qui-gon tp3$ ls -l /dev
total 0
crw-r--r--  1 root root 10,   235 Oct 23 10:36 autofs
drwxr-xr-x  2 root root       140 Oct 23 10:36 block
drwxr-xr-x  2 root root            80 Oct 23 10:36 bsg
crw-------  1 root root 10,   234 Oct 23 10:36 btrfs-control
drwxr-xr-x  3 root root            60 Oct 23 10:36 bus
drwxr-xr-x  2 root root       3860 Oct 24 10:46 char
crw-------  1 root root       5,    1 Oct 23 10:36 console
lrwxrwxrwx  1 root root            11 Oct 23 10:36 core -> /proc/kcore
drwxr-xr-x  2 root root            60 Oct 23 10:36 cpu
crw-------  1 root root 10,   60 Oct 23 10:36 cpu_dma_latency
crw-------  1 root root 10,   203 Oct 23 10:36 cuse
drwxr-xr-x  6 root root       120 Oct 23 10:36 disk
drwxr-xr-x  3 root root       100 Oct 23 10:36 dri
...
```

Agregar un manejador al sistema significa registrarlo con el kernel, al realizar esta operación el kernel asigna un nro (Major number) que identifica al manejador, ver /proc/devices
Para registrar el dispositivo se utiliza la función **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 práctico se puede encontrar en:
https://static.lwn.net/images/pdf/LDD3/ch03.pdf
http://tldp.org/LDP/lkmpg/2.6/html/index.html