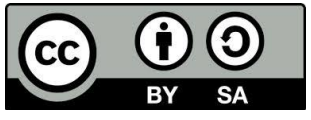
Práctica 1: Módulos

Arquitectura Interna de Linux - 2016



J.C. Sáez
High Performance Computing Division
ArTeCs Group
Universidad Complutense de Madrid

Contenido

Práctica 1: Módulos

Objetivos

1. Familiarizarse con las siguientes abstracciones de Linux:
   - Módulos
   - Sistema de ficheros /proc
   - Listas enlazadas en el kernel
   - Gestión básica de memoria dinámica en el kernel
2. Afrontar las dificultades de la programación en espacio de kernel

ArTeCS

Contenido

Ejercicios I

Ejercicio 1

- `printk()` es un mecanismo de logging. 8 niveles de prioridad (`<linux/kernel.h>`)
  - ¿Qué diferencia encuentras entre KERN_INFO y KERN_ALERT?

Ejercicio 2

- La función de carga de los módulos de ejemplo devuelve 0 ¿qué ocurre cuando se devuelve un número negativo?

Ejercicios II

Ejercicio 3

- Estudiar la implementación del módulo 'clipboard', que exporta una entrada /proc
  - Al cargar/descargar el módulo se creará/eliminará una entrada clipboard en el sistema de ficheros virtual /proc
  - La entrada clipboard puede emplearse como un portapapeles (*clipboard*) del sistema
  - Realizar lecturas y escrituras sobre la entrada /proc/clipboard para entender el comportamiento del módulo
    - `echo hola > /proc/clipboard`
    - `cat /proc/clipboard`

Ejercicios III

Ejercicio 4

- Analizar la implementación del módulo del kernel `modleds.c` que interactúa con el driver de teclado de un PC para encender/apagar los LEDs
  - Al cargar el módulo se encienden los tres leds del teclado y al descargarlo se apagan
- Advertencia: No usar una shell SSH para cargar el módulo. Usar una ventana de terminal en la propia máquina virtual.

Ejercicios IV

Ejercicio 4 (cont.)

- Se ha de prestar especial atención a las siguientes funciones:
  - `get_kbd_driver_handler()`: Se invoca durante la carga del módulo para obtener un puntero al manejador del driver de teclado/terminal
  - `set_leds(handler,mask)`: Permite establecer el valor de los leds. Acepta como parámetro un puntero al manejador del driver y una máscara de bits que especifica el estado de cada LED.

Ejercicios V

Ejercicio 4 (cont.)

- Significado de la máscara de bits de `set_leds()` (parámetro `mask`)
  - bit 0: scroll lock ON/OFF
  - bit 1: num lock ON/OFF
  - bit 2: caps lock ON/OFF
  - bit 3-31: se ignoran
- En cada bit..
  - Si 1 $\rightarrow$ LED ON
  - Si 0 $\rightarrow$ LED OFF

Contenido

Especificación de la práctica

Dos partes

- **Parte A**: Desarrollar un módulo del kernel que permita gestionar los LEDs del teclado
  - El usuario podrá modificar el estado de los LEDs del teclado escribiendo una cadena de caracteres en el fichero `/proc/leds`
- **Parte B**: Crear un módulo `modlist` que gestione una lista enlazada de enteros
  - El usuario podrá realizar operaciones sobre esa lista enlazada escribiendo una cadena de caracteres en la entrada `/proc/modlist`

Parte A: especificación

Parte A

- Crear un módulo del kernel (`procleds.c`) que controle los *leds* del teclado de un PC
  - Reutilizar código del ejemplo `modleds.c`
- El nuevo módulo del kernel expondrá los LEDs al usuario mediante una entrada `/prod/leds`
  - Este fichero especial se creará automáticamente por el módulo al cargarlo y se deberá destruir al descargarlo

Parte A: especificación (cont.)

- El usuario escribirá una cadena de caracteres en el fichero especial `/proc/leds` para encender y apagar LEDs:
  - Cadena puede incluir los caracteres '1','2' y '3'
    - Si aparece el '1' → encender `Num Lock`
    - Si aparece el '2' → encender `Caps Lock`
    - Si aparece el '3' → encender `Scroll Lock`

Comando	Num Lock	Caps Lock	Scroll Lock
`sudo echo 1 > /proc/leds`	ON	OFF	OFF
`sudo echo 123 > /proc/leds`	ON	ON	ON
`sudo echo 32 > /proc/leds`	OFF	ON	ON
`sudo echo > /proc/leds`	OFF	OFF	OFF
`sudo echo 22 > /proc/leds`	OFF	ON	OFF

Parte A: Ejemplo de ejecución

terminal

```
kernel@debian:p1$ sudo insmod procleds.ko
kernel@debian:p1$ sudo echo 12 > /proc/leds
<< Se deberían encender los dos LEDs de más a la izquierda>>
kernel@debian:p1$ sudo echo 3  > /proc/leds
<< Se debería encender solamente el LED de la derecha >>
kernel@debian:p1$
```

Operaciones a nivel de bit

- Encender los leds adecuados implica transformar la cadena escrita por el usuario en una máscara de bits (segundo parámetro de la función `set_leds()`)

Operaciones a nivel de bit en C

```
unsigned int mask=0; /* Número de 32 bits (todo ceros) */

...

/* Poner bit 3 a '1': (desplazamiento a la izquierda y OR) */
mask|=(1<<3);

/* Poner bit 7 a '1' */
mask|=(1<<7);

/* Poner bit 3 a '0': (desplazamiento a la izquierda, NOT y AND) */
mask&=~(1<<3);

/* Comprobar si bit 4 está activo */
if (mask & (1<<4))
    printk(KERN_INFO "Bit 4 está a 1\n");
```

Parte B: Especificación

- Crear un módulo `modlist` que gestione una lista enlazada de enteros

```
struct list_head mylist; /* Lista enlazada */

/* Nodos de la lista */
typedef struct {
        int data;
        struct list_head links;
}list_item_t;
```

- El módulo permitirá al usuario insertar/eliminar elementos de la lista mediante la entrada `/proc/modlist`
  - Cuando el módulo se cargue/descargue se creará/eliminará dicha entrada
- La memoria asociada a los nodos de la lista debe gestionarse de forma dinámica empleando `vmalloc()` y `vfree()`
  - Al descargar el módulo → liberar memoria si lista no vacía

Parte B: Especificación (cont.)

Operaciones soportadas por el módulo

1. Inserción al final de la lista
   - `echo add 10 > /proc/modlist`
2. Eliminación de la lista
   - `echo remove 10 > /proc/modlist`
   - *Borra todas las ocurrencias de ese elemento en la lista*
3. Impresión por pantalla de la lista
   - `cat /proc/modlist`
4. Borrado de todos los elementos de la lista
   - `echo cleanup > /proc/modlist`

Procesando cadena de entrada en `write`

Pseudocódigo write *callback*

```
#define MAX_SIZE_KBUF 100

static ssize_t proc_modlist_write(struct file *filp, const char __user
      *buf, size_t len, loff_t *off) {
    int kbuf[100];
    int val;

    ...

    ... copiar buf al espacio de kernel (kbuf) y cerrar la cadena ...

    if(sscanf(kbuf,"add %i",&val)==1) {
        ... insertar en lista ...
    } else if (sscanf(kbuf,"remove %i",&val)==1) {
        ... eliminar de lista ...
    } else if (...) {
        ...
    }

    return ??;
}
```

Ejemplo de ejecución

terminal

```
kernel@debian$ sudo insmod modlist.ko
kernel@debian$ cat /proc/modlist
kernel@debian$ echo add 10 >  /proc/modlist
kernel@debian$ cat /proc/modlist
10
kernel@debian$ echo add 4 >  /proc/modlist
kernel@debian$ echo add 4 >  /proc/modlist
kernel@debian$ cat /proc/modlist
10
4
4
kernel@debian$ echo add 2 >  /proc/modlist
kernel@debian$ echo add 5 >  /proc/modlist
kernel@debian$ cat /proc/modlist
10
4
4
2
5
```

Ejemplo de ejecución (cont..)

terminal

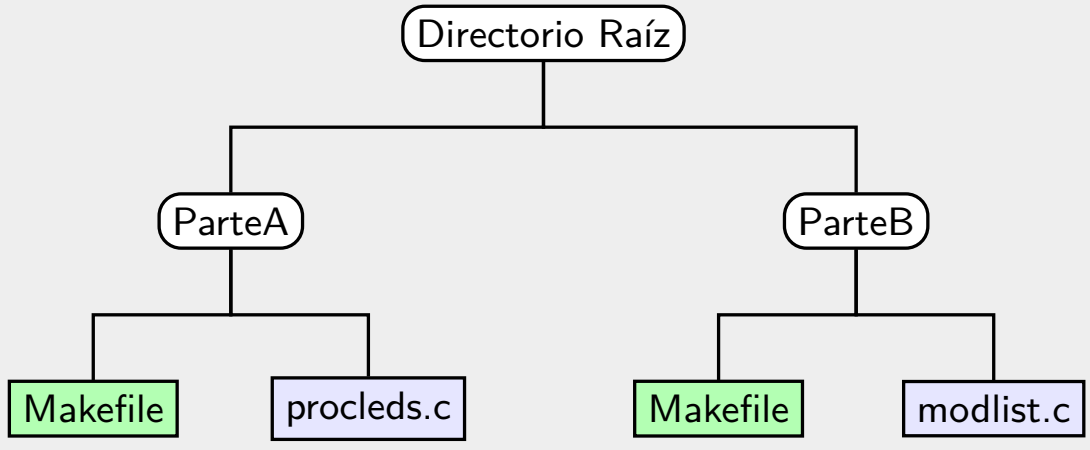
```
kernel@debian$ echo remove 4 > /proc/modlist
kernel@debian$ cat /proc/modlist
10
2
5
kernel@debian$ echo cleanup > /proc/modlist
kernel@debian$ cat /proc/modlist
kernel@debian$
```

ArTeCS

Entrega de la práctica

- Mostrar la práctica funcionando al profesor durante el curso
- Entregar código a través del Campus Virtual

Estructura entrega (en un fichero comprimido .tar.gz o .zip)



Licencia

Arquitectura Interna de Linux - Práctica 1: Módulos
Versión 0.3

Este documento (o uno muy similar) está disponible en
`https://cv4.ucm.es/moodle/course/view.php?id=70009`