Práctica 2: Llamadas al sistema

Juan Carlos Sáez Alcaide

Índice

1 Introducción y objetivos

El principal objetivo de esta práctica es familiarizarse con la implementación de llamadas al sistema en Linux y su procedimiento de invocación, así como con la compilación del kernel Linux.

Debido a la naturaleza de esta práctica no será necesario desarrollar módulos cargables, sino realizar modificaciones en el kernel Linux v5.10.45 (versión instalada en la máquina virtual de la asignatura). Como fase preparatoria para los proyectos de desarrollo descritos en la sección 3, es preciso realizar previamente una serie de ejercicios sencillos que se describen a continuación.

Nota importante: Para realizar el Ejercicio 3 y la parte B de la práctica en el laboratorio 8 será preciso tomar prestado un teclado USB del personal de laboratorios. El préstamo puede realizarse al comienzo de cada sesión, y solo es necesario en los laboratorios 8 y 9 donde los LEDs del teclado de los puestos no funcionan correctamente debido a la compleja configuración del teclado (no se conecta directamente por USB al equipo). En el resto de laboratorios de la Facultad de Informática, todo debería funcionar correctamente con el teclado instalado en el puesto.

2 Ejercicios

Ejercicio 1

Estudiar la implementación del programa `cpuinfo.c` suministrado entre los ficheros de la práctica. Este programa imprime por pantalla el contenido del fichero especial `/proc/cpuinfo` haciendo uso de las llamadas al sistema

`open()` y `close()`, y las funciones `printf()` y `syscall()`. La entrada `/proc/cpuinfo` permite obtener información acerca de las CPUs o procesadores lógicos disponibles en el sistema:

```
kernel@debian:~$ cat /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 23
model name  : Intel(R) Xeon(R) CPU           E5450  @ 3.00GHz
stepping    : 10
cpu MHz     : 2003.000
cache size  : 6144 KB
physical id : 0
siblings    : 4
core id     : 0
cpu cores   : 4
apicid      : 0
initial apicid  : 0
fpu     : yes
fpu_exception   : yes
cpuid level : 13
wp      : yes
...
```

¿Qué llamada al sistema invoca el programa `cpuinfo.c` mediante `syscall()`?

Reescribir el programa suministrado reemplazando las llamadas a `open()`, `close()` y `printf()` por invocaciones a `syscall()` que tengan el mismo comportamiento.

Ejercicio 2

Leer el artículo "Introducción a bpftrace" que se encuentra disponible aquí. En dicho artículo, se muestran comandos de ejemplo de uso de la herramienta. Estos comandos deben ejecutarse en la máquina virtual de Debian de la asignatura para familiarizarse con `bpftrace`.

Se aconseja encarecidamente poner en práctica lo aprendido sobre `bpftrace` en la depuración de las prácticas sobre la máquina virtual.

Ejercicio 3

Estudiar el módulo del kernel `modleds.c` que interactúa con el driver de teclado de un PC para encender/apagar los leds num-lock, caps-lock y scroll-lock. Al cargar el módulo se encienden los tres leds del teclado y al descargarlo se apagan. **Nota importante**: El módulo debe cargarse desde una ventana de terminal dentro de la propia máquina virtual, no desde un shell SSH.

En el código del ejemplo 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. Cada bit controla un led específico del teclado:
  - bit 0: scroll lock ON/OFF
  - bit 1: num lock ON/OFF
  - bit 2: caps lock ON/OFF
  - bits 3-31: se ignoran

En cada bit un valor "1" enciende el LED correspondiente, y un "0" lo apaga.

3 Desarrollo de la práctica

Esta práctica consta de dos partes: A y B.

3.1 Parte A

En esta parte de la práctica se ha de crear la llamada al sistema `lin_hello` ("Hola Mundo") siguiendo las instrucciones proporcionadas en el tema "Llamadas al Sistema". También ha de crearse un parche del kernel con las modificaciones realizadas.

Una vez se haya hecho funcionar el código de esta práctica guiada, se ha de mostrar al profesor (1) el funcionamiento de la llamada al sistema en el laboratorio y (2) el parche del kernel creado.

3.2 Parte B

Crear una llamada al sistema `ledctl()`, que permita a los programas de usuario encender y apagar los LEDs del teclado del PC. Esta llamada se implementará reutilizando el código del módulo de ejemplo `modleds.c` (Ejercicio 3). Además de la llamada al sistema se ha de implementar un programa de usuario `ledctl_invoke` que permita invocar la llamada `ledctl()` desde línea de comandos.

3.2.1 Especificación de `ledctl()`

La llamada `ledctl()` tendrá la siguiente especificación:

```
long ledctl(unsigned int leds);
```

El parámetro `leds` es una máscara de bits que indica qué LEDs se encenderán/apagarán:

- bit 2: encender/apagar *Num Lock*
- bit 1: encender/apagar *Caps Lock*
- bit 0: encender/apagar *Scroll Lock*

La llamada al sistema retornará 0 en caso de éxito o -1 en caso de fallo. Uno de los posibles casos de error es que el usuario pase un parámetro que no sea 0-7. Nótese que la implementación en sí de la llamada en el kernel devolverá –en caso de error– un número negativo que codifica dicho error. Cabe también destacar que si se produce un error, la función `syscall()` –a usar en el programa `lectl_invoke`– ya se encarga de devolver -1 al programa de usuario, y de almacenar el código de error en la variable global `errno`.

La siguiente tabla muestra qué efecto tendrá en el estado de los leds el uso de distintos valores pasados como argumento a la llamada al sistema `ledctl()`:

Parámetro de `ledctl()`	Num Lock	Caps Lock	Scroll Lock
4	ON	OFF	OFF
7	ON	ON	ON
3	OFF	ON	ON
0	OFF	OFF	OFF
2	OFF	ON	OFF

Para implementar la llamada al sistema dentro del kernel (*handler function*), se debe utilizar la macro `SYSCALL_DEFINE1()` como sigue:

```
#include <linux/syscalls.h> /* For SYSCALL_DEFINEi() */
#include <linux/kernel.h>
```

```
SYSCALL_DEFINE1(ledctl,unsigned int,leds)
{
    /* To be completed ... */

    return 0;
}
```

3.2.2 Programa `ledctl_invoke`

Para llevar a cabo la depuración de la llamada al sistema `ledctl()` y poder invocarla desde un terminal se desarrollará el programa de usuario `ledctl_invoke`, cuyo modo de uso de describe a continuación:

```
$ ledctl_invoke <argumento_de_ledctl>
```

Ejemplo: `$ ./ledctl_invoke 4`

El programa de usuario debe gestionar errores de forma adecuada, asegurando que (1) el primer argumento es un número y está en el rango 0-7, y (2) que en el caso de que `ledctl()` devuelva un error, el mensaje de error correspondiente se muestre con `perror()`.

Si el programa se codifica en un fichero `ledct_invoke.c`, el ejecutable se puede generar invocando `gcc` como sigue:

```
$ gcc -Wall -g ledctl_invoke.c -o ledct_invoke
```

3.2.3 Consejos para la implementación de la llamada al sistema

La implementación de cualquier llamada al sistema requiere modificar el kernel. En particular, por cada fallo detectado al desarrollar `ledctl()`, es preciso realizar los siguientes pasos:

1. Realizar modificaciones pertinentes en el código del kernel para subsanar el error
2. Compilar el kernel modificado (la configuración solo se hace la primera vez)
3. Instalar paquetes (*image* y *headers*) en la máquina virtual y reiniciar
4. Probar código usando programa `ledctl_invoke` (a desarrollar)
5. Si fallo, ir a 1. En otro caso, hemos acabado :-)

Como este proceso puede llegar a ser lento y tedioso, se aconseja desarrollar un módulo del kernel auxiliar para depurar el código de la llamada al sistema antes de introducir su implementación en el kernel. Por ejemplo, el módulo de depuración podría exportar una entrada /proc de sólo escritura que permita modificar el estado de los leds al escribir en ella un entero en el rango 0-7. Si se hace esto la escritura del número 5 en dicha entrada debería encender solamente los dos leds de los extremos:

```
$ sudo echo 0x5 > /proc/ledctl
```

3.2.4 Ejemplo de ejecución

A continuación se muestra una secuencia de comandos de prueba que pueden emplearse tras arrancar el sistema con el kernel modificado –con la llamada`ledctl()` ya implementada–:

```
kernel@debian:p2$ gcc -g -Wall ledctl_invoke.c -o ledctl_invoke
kernel@debian:p2$ ./ledctl_invoke
Usage: ./ledctl_invoke <ledmask>

kernel@debian:p2$ sudo ./ledctl_invoke 0x6
## Se deberían encender los dos LEDs de más a la izquierda

kernel@debian:p2$ sudo ./ledctl_invoke 0x1
```

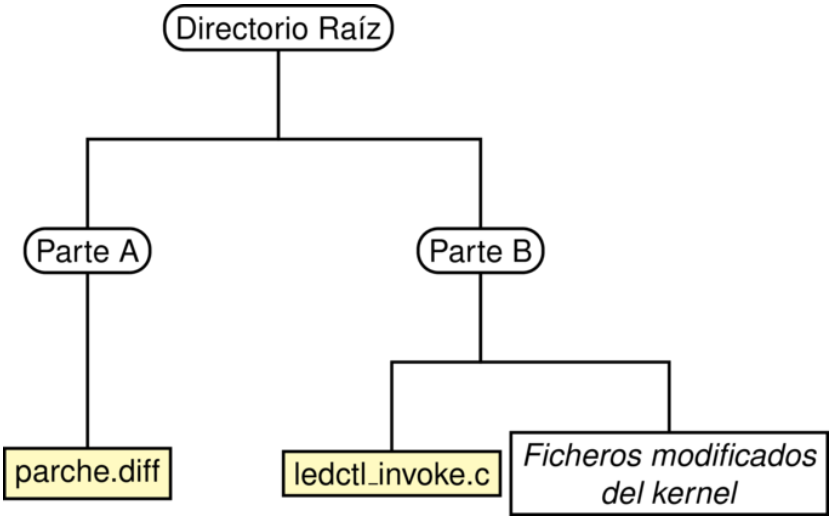
```
## Se debería encender solamente el LED de la derecha

kernel@debian:p2$ sudo ./ledctl_invoke 0x2
## Se debería encender solamente el LED del centro

kernel@debian:p2$
```

3.3 Entrega de la práctica

La práctica ha de entregarse a través del Campus Virtual en un fichero comprimido (.tar.gz o .zip) con la siguiente estructura de directorios:



Plazo de entrega: Hasta el **13 de octubre**.

Es obligatorio mostrar el funcionamiento de la práctica en clase.