



Práctica 1: Módulos

LIN - Curso 2021-2022



Contenido



1 Introducción

2 Ejercicios

3 Introducción a Ftrace

4 Práctica



Contenido



1 Introducción

2 Ejercicios

3 Introducción a Ftrace

4 Práctica



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
 - Depuración con Ftrace
- 2** Afrontar las dificultades de la programación en espacio de kernel

Contenido



1 Introducción

2 Ejercicios

3 Introducción a Ftrace

4 Práctica



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 el mecanismo de paso de parámetros a módulos del kernel
 - `<linux/moduleparam.h>`
 - Ejemplo 2.7 de *Linux Kernel Module Programming Guide* (módulo `hello-5`)

Ejercicio 4

- Los comandos básicos de gestión de módulos (`lsmod`, `insmod`, etc.) no son ejecutables independientes, sino enlaces simbólicos a la utilidad `kmod`.
 - 1 Verifica que estos comandos son en realidad enlaces simbólicos usando `stat`
 - 2 Intenta obtener un listado de los módulos del kernel que están cargados invocando `kmod` en lugar de `lsmod`.
 - Pista: Consultar la salida de `kmod -h`

Ejercicios III



Ejercicio 5

- 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

Contenido



1 Introducción

2 Ejercicios

3 Introducción a Ftrace

4 Práctica



Ftrace

- Herramienta de depuración/inspección del kernel Linux
- Se usa realizando lecturas y escrituras en un conjunto de entradas en el sistema de ficheros *debugfs*
 - Directorio: `/sys/kernel/debug/tracing`
 - Sólo el usuario `root` puede configurar/usar `ftrace`
- Consta de un conjunto de *tracers*
 - `nop`
 - `function`
 - `function_graph`
 - ...
- Documentación de `ftrace`
 - <https://www.kernel.org/doc/Documentation/trace/ftrace.txt>

Montar debugfs para poder usar ftrace

- Las entradas de ftrace sólo estarán accesibles si:
 - 1 Kernel compilado con soporte de ftrace (CONFIG_FTRACE=y)
 - 2 ... y debugfs está montado (en Debian 10 lo está por defecto)
 - `mount -t debugfs nodev /sys/kernel/debug`

Montando debugfs (si no estuviera ya montado)

```
kernel@debian:~$ sudo -i
[sudo] password for kernel:
root@debian:~# ls /sys/kernel/debug/tracing
ls: cannot access /sys/kernel/debug/tracing: No such file or directory
root@debian:~# ls
root@debian:~# mount -t debugfs nodev /sys/kernel/debug
root@debian:~# cd /sys/kernel/debug/tracing
root@debian:/sys/kernel/debug/tracing# ls
available_events          dyn_ftrace_total_info   kprobe_profile
saved_cmdlines           set_ftrace_pid          stack_trace_filter      tracing_cpumask
available_filter_functions enabled_functions        max_graph_depth
saved_cmdlines_size      set_graph_function      trace
tracing_max_latency      available_tracers       events                  options
set_event               set_graph_notrace       trace_clock             tracing_on
...
```



Entradas básicas de Ftrace

■ Entradas básicas en `/sys/kernel/debug/tracing`

Entrada	Descripción
<code>tracing_on</code>	Permite activar/desactivar ftrace o consultar estado actual. Escribir la cadena "1" para activar ftrace o "0" para desactivarlo.
<code>trace</code>	Al leer de esta entrada se muestran los mensajes almacenados en los <i>buffers</i> de ftrace (un <i>buffer</i> por CPU)
<code>trace_pipe</code>	Similar a <code>trace</code> , pero además los buffers se vacían al mostrar su contenido (semántica productor/consumidor)
<code>available_tracers</code>	Lista el conjunto de <i>tracers</i> disponibles
<code>current_tracer</code>	Permite consultar/modificar el <i>tracer</i> activo leyendo/escribiendo en la entrada
<code>available_filters</code>	Lista el conjunto de funciones del kernel o de los módulos cargados que pueden "filtrarse" al usar el <code>tracer function</code> .
<code>set_ftrace_filter</code>	Permite establecer la función (o funciones) para las que ftrace insertará un mensaje en el buffer cuando éstas sean invocadas.

nop tracer y trace_printk()

- *Tracer* por defecto en el kernel
- Captura únicamente los mensajes que el kernel o los módulos imprimen con la función `trace_printk()`

`trace_printk()`

- Para usar `trace_printk()` desde un módulo del kernel ...
 - `#include <linux/ftrace.h>`
- Uso similar a `printf()`, pero mensaje se inserta en buffer interno de ftrace
- Mucho más eficiente que `printk()`. Además, si ftrace está desactivado, no tiene efecto (modo *silencioso*)

Ejemplo de uso de nop tracer (1/4)

- Modificaremos el módulo de ejemplo clipboard para que muestre un mensaje con `trace_printk()` y capturaremos la salida con `ftrace`

Adiciones en clipboard.c (en verde)

```
#include <linux/vmalloc.h>
#include <asm-generic/uaccess.h>
#include <linux/ftrace.h>
...
static ssize_t clipboard_write(struct file *filp, const char __user *buf, size_t len, loff_t
    *off) {
...

    clipboard[len] = '\0'; /* Add the '\0' */
    *off+=len;             /* Update the file pointer */

    trace_printk("Current value of clipboard: %s\n",clipboard);

    return len;
}
...
```



Ejemplo de uso de nop tracer (2/4)

■ Compilar y cargar el módulo

Terminal

```
kernel@debian:/tmp/FicherosP1/Clipboard$ make
make -C /lib/modules/5.10.45-lin/build M=/tmp/FicherosP1/Clipboard modules
make[1]: se entra en el directorio `/usr/src/linux-headers-5.10.45-lin'
  CC [M] /tmp/FicherosP1/Clipboard/clipboard.o
Building modules, stage 2.
MODPOST 1 modules
  CC      /tmp/FicherosP1/Clipboard/clipboard.mod.o
  LD [M] /tmp/FicherosP1/Clipboard/clipboard.ko
make[1]: se sale del directorio `/usr/src/linux-headers-5.10.45-lin'
kernel@debian:/tmp/FicherosP1/Clipboard$ sudo insmod clipboard.ko
[sudo] password for kernel:
kernel@debian:/tmp/FicherosP1/Clipboard$
```

Ejemplo de uso de nop tracer (3/4)

- Abriremos 2 terminales
 - (Primer terminal - root)
 - 1 Asegurarse que ftrace y nop tracer activos
 - 2 Leer de la entrada `trace_pipe` (bloqueante)
 - (Segundo terminal)
 - 1 Escribir la cadena "Test" en la entrada `/proc/clipboard`
 - 2 Escribir la cadena "Something" en la entrada `/proc/clipboard`
- Las acciones realizadas en el segundo terminal harán que se muestren mensajes por el primero (salida de `cat trace_pipe`)

Ejemplo de uso de nop tracer (4/4)

Terminal 1

```
root@debian:/sys/kernel/debug/tracing# cat current_tracer
nop
root@debian:/sys/kernel/debug/tracing# cat tracing_on
1
root@debian:/sys/kernel/debug/tracing# cat trace_pipe
bash-16182 [000] .... 1065.269409: clipboard_write:
Current value of clipboard: Test

bash-16182 [000] .... 1100.023458: clipboard_write:
Current value of clipboard: Something
```

Terminal 2

```
kernel@debian:/tmp/FicherosP1/Clipboard$ echo "Test" > /proc/clipboard
kernel@debian:/tmp/FicherosP1/Clipboard$ echo "Something" > /proc/clipboard
kernel@debian:/tmp/FicherosP1/Clipboard$
```

Tracer function

- Vuelca un “mensaje” en el buffer de ftrace cuando se ejecuta cierta función del kernel
 - Permite ver qué funciones se invocan sin modificar el código del kernel (o de un módulo)
- Soporta filtros de funciones
 - Escribir nombre(s) de funcion(es) a trazar en `set_ftrace_filter`
 - El listado de funciones que pueden seleccionarse se puede obtener leyendo de la entrada `available_filter_functions`
- Por defecto, no hay ningún filtro → ¡¡Se trazan todas las funciones (mucha sobrecarga)!!
 - Aconsejable desactivar temporalmente ftrace (`tracing_on`) hasta que se establezcan correctamente los filtros de funciones

Ejemplo de uso del tracer function (1/2)

- Usaremos ftrace para que nos avise cuándo se invoca la función `clipboard_read()` del módulo `clipboard`
 - No es preciso modificar el código para esto

Pasos (desde `/sys/kernel/debug/tracing` como root)

- 1 Desactivar temporalmente ftrace
`$ echo 0 > tracing_on`
- 2 Activar function tracer y comprobar que se activó correctamente:
`$ echo function > current_tracer ; cat current_tracer`
- 3 Preparar filtros de ftrace
`$ echo clipboard_read > set_ftrace_filter`
- 4 Activar ftrace
`$ echo 1 > tracing_on`

Ejemplo de uso del tracer function (2/2)

- Una vez configurado el tracer function, abrir 2 terminales
 - (Primer terminal - root)
 - Leer de la entrada trace_pipe (bloqueante)
 - (Segundo terminal)
 - Leer de la entrada /proc/clipboard

Terminal 1

```
root@debian:/sys/kernel/debug/tracing# cat trace_pipe
cat-16406 [000] .... 3166.842845: clipboard_read <-proc_reg_read
cat-16406 [000] .... 3166.844485: clipboard_read <-proc_reg_read
```

Terminal 2

```
kernel@debian:/tmp/FicherosP1/Clipboard$ cat /proc/clipboard
Something
kernel@debian:/tmp/FicherosP1/Clipboard$
```

¿Por qué clipboard_read() se invoca 2 veces?

Contenido



1 Introducción

2 Ejercicios

3 Introducción a Ftrace

4 Práctica



Especificación de la práctica

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

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

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

- 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

Especificación de la práctica

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`

Se aconseja utilizar `sscanf()` para procesar los comandos del usuario

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$
```

Partes opcionales

- **(Opcional 1)** Modificar el módulo de la práctica para que la lista gestionada sea de cadenas de caracteres
 - La memoria de las cadenas debe reservarse con `vmalloc()`
 - **NOTA:** Se valorará positivamente la inclusión de sentencias de compilación condicional para mantener en un mismo fichero fuente las implementaciones del módulo con lista de enteros (básica) y lista de cadenas de caracteres (opcional)
 - Los símbolos de preprocesador se especifican con la opción `-D`, y a través de la variable de entorno `EXTRA_CFLAGS`
 - Ej. compilación: `make EXTRA_CFLAGS=-DPARTE OPCIONAL`

```
#ifdef PARTE OPCIONAL
... Fragmento de código específico para lista de cadenas...
#else
... Fragmento de código específico para lista de enteros...
#endif
```

Partes opcionales (cont.)

- **(Opcional 2)** Reimplementar la read callback de la entrada /proc empleando `seq_printf()` mediante la abstracción de `seq_files` de Linux
 - Aconsejable mantener un contador con el número de elementos de la lista

Ventajas del uso de `seq_file`:

- Soporte especial para recorrido de secuencias de elementos
- Gestión implícita del buffer de usuario
 - No requiere usar `copy_to_user()` ni `sprintf()`

Parte opcional 2 (documentación)

Documentación sobre seq_files para Linux 5.7.x y anteriores...

- Sección 10.2 “Professional Linux Kernel Architecture”
- The seq_file Interface (kernel docs)

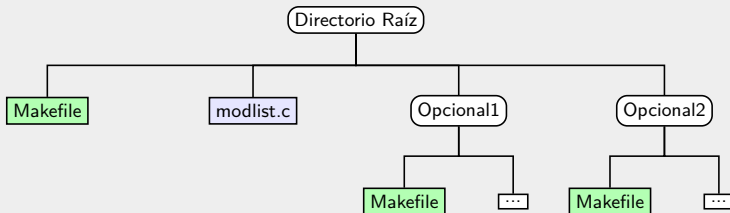
Ejemplo implementación para Linux 5.10.x de entrada /proc/cpuinfo

- <https://elixir.bootlin.com/linux/v5.10.60/source/fs/proc/cpuinfo.c#L12>
- <https://elixir.bootlin.com/linux/v5.10.60/source/arch/x86/kernel/cpu/proc.c#L177>
- **Cambios con respecto a versiones previas del kernel:**
 - 1 Usar struct proc_ops en lugar de struct file_operations para entrada /proc
 - 2 Se ha de definir la callback read_iter en lugar de read en struct proc_ops

Entrega de la práctica

- A través del Campus Virtual
 - Hasta el 1 de octubre
- Aconsejable mostrar el funcionamiento antes de hacer la entrega

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



*This work is licensed under the Creative Commons **Attribution-Share Alike 3.0 Spain License**. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/es/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.*

*Esta obra está bajo una licencia **Reconocimiento-Compartir Bajo La Misma Licencia 3.0 España de Creative Commons**. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.*

Este documento (o uno muy similar) está disponible en
<https://cvmdp.ucm.es/moodle/course/view.php?id=20152>

