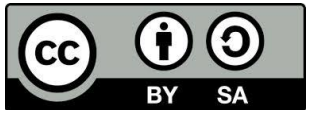
Práctica 2: Syscalls

Arquitectura Interna de Linux - 2016



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

Contenido

Contenido

Práctica 2: Llamadas al sistema

Objetivos

- Familiarizarse con:
  - Implementación de llamadas al sistema en Linux y su procedimiento de invocación
  - Compilación del kernel Linux
  - Creación de parches

ArTeCS

Contenido

Ejercicios

Ejercicio 1

- Estudiar la implementación del programa cpuinfo.c
  - Este programa imprime por pantalla el contenido de `/proc/cpuinfo` haciendo uso de las llamadas al sistema `open()` y `close()`, y las funciones `printf()` y `syscall()`.
  - ¿Qué llamada al sistema invoca el programa mediante `syscall()`?
  - Reescribir el programa anterior reemplazando las llamadas a `open()`, `close()` y `printf()` por invocaciones a `syscall()` que tengan el mismo comportamiento.

Ejercicios

- La entrada /proc/cpuinfo permite obtener información acerca de las CPUs del sistema

Terminal

```
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
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
...
```

Contenido

Partes de la práctica

(Parte A.) Crear llamada al sistema "Hola Mundo" (`lin_hello`)

- Seguir instrucciones del tema "Llamadas al Sistema"
- Enseñar funcionamiento al profesor en el laboratorio
- Crear un parche con los cambios realizados

(Parte B. - Opcional) Implementar llamada al sistema `ledctl()`

1. La llamada permitirá que los programas de usuario puedan encender/apagar los LEDs del teclado
   - Exige modificar el kernel para incluir llamada al sistema `ledctl()`
2. Además se ha de implementar el programa de usuario `ledctl_invoke` que permita invocar la llamada al sistema desde terminal

Especificación de `ledctl()` (I)

Llamada al sistema `ledctl()`

```
int ledctl(const char* command);
```

- **Parámetro**: Cadena de caracteres que especifica qué LEDs se encenderán/apagarán
- **Valor de retorno**: 0 en caso de éxito; -1 en caso de fallo
  - Advertencia: La implementación en sí de la llamada (kernel) devolverá un número negativo que codifica el error
    - En caso de error la libc devolverá -1 al programa de usuario, y el código de error quedará almacenado en la variable global `errno`

Especificación de `ledctl()` (II)

Formato parámetro ledctl

- `ledctl()` acepta como parámetro una cadena de caracteres formada por números del 0 al 3
  - 0 → apagar los tres leds
  - 1 → encender *Num Lock*
  - 2 → encender *Caps Lock*
  - 3 → encender *Scroll Lock*

Cadena	Num Lock	Caps Lock	Scroll Lock
"1"	ON	OFF	OFF
"123"	ON	ON	ON
"32"	OFF	ON	ON
"0"	OFF	OFF	OFF
"22"	OFF	ON	OFF

Programa `ledctl_invoke`

- Para llevar a cabo la depuración de la llamada al sistema se desarrollará un programa de usuario para invocarla desde terminal
  - En caso de que `ledctl()` devuelva un error, el programa mostrará el error correspondiente con `perror()`
- Modo de uso

  ```
  $ ledct_invoke <comando_ledctl>
  ```

  - Ejemplo: $ ./ledctl_invoke “13”
    - invocará `ledctl("13");`
  - Para compilar `ledct_invoke.c` invocaremos al compilador gcc directamente:

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

Implementación Parte B (I)

- La implementación de la llamada al sistema requiere modificar el kernel
  - Por cada fallo detectado:
    1. Modificar código del kernel
    2. Compilar y reinstalar kernel
    3. Reiniciar la máquina
- Se aconseja reutilizar el código desarrollado en la practica 1A para implementar llamada al sistema

Implementación Parte B (II)

Pasos a seguir

1. Realizar modificaciones pertinentes en el código del kernel
2. Compilar el kernel modificado
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 :-)

ArTeCS

Implementación Parte B (III)

- Al definir la llamada al sistema dentro del kernel, se debe utilizar la macro `SYSCALL_DEFINE1()`

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

SYSCALL_DEFINE1(ledctl,const char*,command)
{
    int kbuf[MAX_SIZE_KBUF]

    /** copiar command al espacio de kernel **/
    if (strncpy_from_user(kbuf,command,MAX_SIZE_KBUF)<0)
        return -EFAULT;

    ...

    Procesar cadena de caracteres (kbuf) del usuario

    Modificar los leds adecuadamente, enviando solicitud al driver de
        teclado

    return 0;
}
```

Parte B: Ejemplo de ejecución

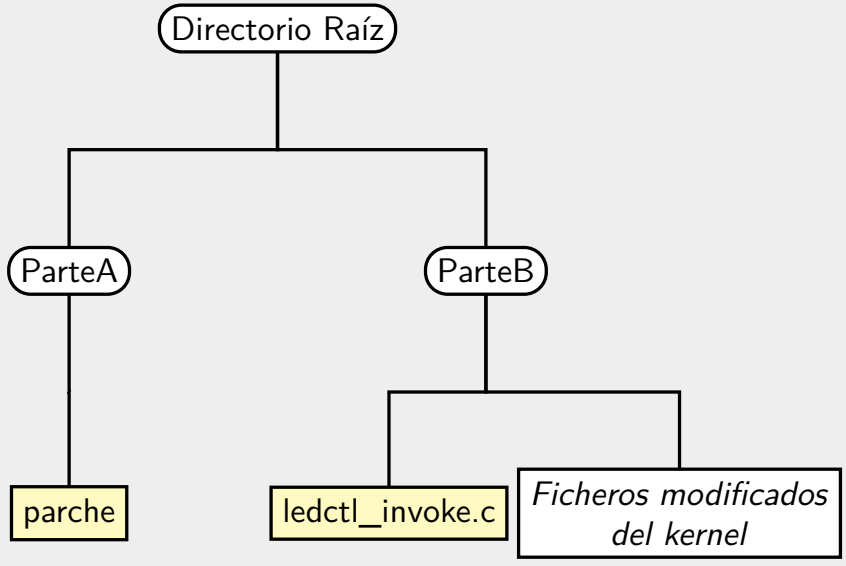
- Arrancar la MV con el kernel modificado con `ledctl()` y abrir una ventana de terminal...

```
terminal
kernel@debian:p2$ gcc -g -Wall ledctl_invoke.c -o ledctl_invoke
kernel@debian:p2$ ./ledctl_invoke
Usage: ./ledctl_invoke <command>
kernel@debian:p2$ sudo ./ledctl_invoke 12
<< Se deberían encender los dos LEDs de más a la izquierda>>
kernel@debian:p2$ sudo ./ledctl_invoke 3
<< Se debería encender solamente el LED de la derecha >>
kernel@debian:p2$
```

Entrega de la práctica

- Mostrar la práctica funcionando al profesor durante el curso
  - Parte opcional puede hacerse al finalizar 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 2: Syscalls
Versión 0.3

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