UNIVERSIDAD DE ALCALÁ

Departamento de Automática

Grado en Ingeniería Informática

Práctica opcional: Añadir una nueva llamada al sistema en el núcleo de Linux

Sistemas Operativos

Índice

1. **Competencias asociadas al ejercicio opcional** **3**

2. **Introducción** **3**

3. **Procedimiento para añadir una llamada al sistema en Linux** **5**

4. **Testear la nueva llamada al sistema** **7**

5. **Ejercicio** **8**

1. Competencias asociadas al ejercicio opcional

1. Ser capaz de comprender qué es una llamada al sistema.
2. Ser capaz de comprender el modo de petición de un servicio al sistema operativo o llamada al sistema.
3. Ser capaz de comprender cómo es el mecanismo de llamadas al sistema.
4. Ser capaz de distinguir los servicios POSIX de las llamadas al sistema y la relación entre ellas.
5. Ser capaz de distinguir los pasos en la ejecución de un servicio POSIX y el modo de ejecución en cada uno de ellos (usuario o supervisor) y el espacio en el que se realizan (espacio de usuario y espacio de núcleo).
6. Ser capaz de expresar con claridad razonando sobre aspectos relativos a la ejecución de una llamada al sistema en Linux.

2. Introducción

El sistema operativo proporciona a los programas de usuario una interfaz que consiste en un conjunto de rutinas denominadas *llamadas al sistema*. Estas funciones permiten que el sistema operativo ofrezca a los procesos un acceso controlado a los recursos del sistema (archivos, procesos, etc.).

Los procesos de usuario (que pueden ser órdenes ejecutadas por él), ejecutados en espacio de usuario, se comunican con el sistema operativo y le solicitan servicios a través de las llamadas al sistema.

A cada llamada del sistema le corresponde, *en general*, una función, en lenguaje C, de una biblioteca (*libc*)[1] situada en el espacio de direcciones del usuario, fuera del núcleo del sistema operativo, que los programas de usuario pueden invocar (véase Figura 1). Estas funciones en lenguaje de alto nivel se encargan de pasar al sistema operativo los parámetros de la llamada del sistema recibidos, normalmente, a través de la pila (por ejemplo, en Linux, el mecanismo de paso de parámetros se realiza a través de los registros de la máquina previamente a la invocación de la llamada al sistema específica), y después, realiza la invocación a la instrucción TRAP (*evento síncrono*) y recibe el código de retorno (si ha habido error, el código de retorno es -1 y asigna normalmente el tipo de fallo concreto en la variable global `errno`).

En concreto, en Linux, a las funciones de biblioteca fuera del núcleo se las denominada, por su propia naturaleza, rutinas “wrapper”. Uno de sus propósitos fundamentales es ocultar los detalles de la interfaz de las llamadas al sistema del núcleo para facilitar la programación al usuario; abstrae los detalles hardware de la llamada al sistema (qué tipo de instrucción TRAP es necesaria para realizar la llamada al sistema según la plataforma, modo de pasar parámetros al sistema operativo, etc).

[1] Sin embargo, una función en C de esta biblioteca no tiene por qué “envolver” biunívocamente una llamada al sistema; dos funciones de la biblioteca pueden implicar la misma) por ejemplo, `waitpid()` y `wait()` invocan una misma llamada al sistema), o puede no implicar inclusive ninguna llamada al sistema.

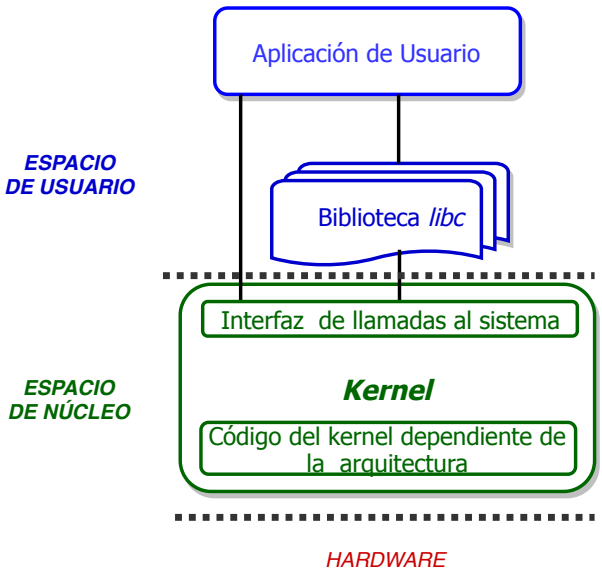

Figura 1: Interfaz de del sistema operativo con el usuario.

Recuerde que la instrucción TRAP es una instrucción específica, según el ensamblador concreto de cada plataforma hardware y según el sistema operativo. Al igual que cualquier tipo de evento, la ejecución de esta instrucción TRAP provoca el cambio de modo de ejecución de la CPU; de modo usuario a modo supervisor (o kernel) y toma el control el sistema operativo. Ya en modo supervisor, el sistema operativo chequea primero los parámetros de la llamada al sistema para testear su validez y, si es correcta, invoca a la rutina que realiza la funcionalidad o servicio concreto asociado a la llamada al sistema (el flujo de ejecución completo en la ejecución de una llamada al sistema puede verse en (**?**)).

En muchas ocasiones, a las funciones de biblioteca se les denomina directamente llamadas al sistema. Como se ha podido concluir de lo anterior, estando estrechamente relacionados ambos conceptos, no es lo mismo. Una vez que comprenda el mecanismo de llamadas al sistema, en la siguiente práctica comenzará a programar ya utilizando las funciones de la interfaz API (`Application Programming Interface`) del estándar POSIX (*P*ortable *O*perating *S*ystem *I*nterface for Uni*X*) de IEEE, cuyos archivos objeto están contenidos en la biblioteca *libc* de Linux.

A modo de ejemplo, la llamada al sistema `sys_read()` en Linux tiene asociada una función de biblioteca del estándar POSIX denominada `read()`. Para utilizar esta función en un programa fuente en C, se le deben pasar tres parámetros: el primero especifica el archivo a leer (descriptor de archivo devuelto por el sistema operativo al abrir el archivo, previamente a la operación de lectura, si todo ha ido bien en la apertura), el segundo un *búffer* para almacenar en él los datos leídos del archivo y el tercero representa los bytes que se pretenden leer. Su prototipo es el siguiente:

```
ssize_t read (int fd, void * buf, size_t count);
```

Al invocar a la función `read()` (que envuelve o conlleva en su definición la llamada al sistema `sys_read ()`), los datos del archivo cuyo descriptor de archivo es `fd` (identifi-

cador, número entero, del archivo), se copian en `buf`. La función devuelve el número de bytes realmente leídos; normalmente este número es el mismo que `count` (los bytes que se desean leer), pero puede ser menor si, por ejemplo, se encuentra el final del archivo mientras se lee. Si la llamada al sistema no puede finalmente realizarse, bien a causa de un parámetro no válido, bien por un error de disco, etc., el valor de retorno se establece en -1, y el número de error concreto se coloca en la variable global `errno`, que, desde el programa, puede ser accedida para proporcionar más información sobre el fallo. Los programas siempre deben verificar los resultados de estas funciones que envuelven las llamadas al sistema para ver si se produjeron errores.

Para más detalles sobre el mecanismo de llamadas al sistema en Linux se sugiere al alumno (**?**).

El objetivo de esta práctica opcional es que el estudiante pueda comprender en profundidad el flujo de control y las estructuras manejadas durante la ejecución de una llamada al sistema, interfaz de comunicación entre el usuario (bien cuando trabaja con la interfaz de usuario -órdenes- o con la interfaz de aplicaciones -programas-) y el sistema operativo. Se incidirá también en la estrecha relación entre llamadas al sistema y servicios POSIX y cómo el sistema operativo realiza de forma controlada el acceso a los recursos ejecutando en modo núcleo el servicio proporcionado por la llamada al sistema. Para ello, se pretende que el alumno realice un ejercicio práctico sencillo: añadir una sencilla llamada al sistema al núcleo de Linux.

3. Procedimiento para añadir una llamada al sistema en Linux

Para comprender a fondo el mecanismo de las llamadas al sistema en Linux, se detallan a continuación los pasos necesarios para añadir una llamada al sistema en el núcleo de Linux (**?**):

1. Instale, si no lo ha hecho ya, la distribución Ubuntu 18.10, u otra versión estable, de 64-bits (si es el caso de su plataforma) en una máquina virtual (VM). Se recomienda configurar la VM con todos los núcleos del procesador para que la posterior compilación del kernel de Linux sea más rápida. Además, aumente al máximo el espacio de almacenamiento de la VM. Compruebe en la consola de la VM la versión de Ubuntu y del kernel (use la orden `man` para localizar el parámetro adecuado del comando `uname`).

2. Instale, si no lo tuviera: *build-essential* (herramientas de desarrollo). Use el comando `apt-get` para instalar el paquete. Tras ello, `sudo apt-get update` y `sudo apt-get upgrade`.

3. Descargue, con ayuda del comando `wget`, el código fuente del kernel de Linux -versión 4.17.x más reciente-[2].

4. Desempaquete en un directorio el código fuente descargado. Se recomienda realizar esta tarea en `/usr/src`. Para desempaquetar, use el comando `tar` y los parámetros correspondientes para ello vistos en la práctica 1 (si no recuerda los parámetros use la orden `man` o busque en Internet).

[2] `wget` soporta *http*, *https* y *ftp*: `wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.17.x.tar.gz`

5. Compruebe que, efectivamente, en el directorio donde ha desempaquetado el código fuente del kernel hay un directorio relativo a esa versión y acceda a ese directorio.

6. Cree en el directorio al que ha accedido en el paso anterior una carpeta `practicaLab` y acceda también a ella.

7. Edite un archivo `saying.c` con el siguiente contenido. Con ello se pretende crear la nueva llamada al sistema (debe estar precedida por `asmlinkage` en su cabecera y un prefijo "sys_", como puede observar, en su nombre):

```
1 #include <linux/kernel.h>
2
3 asmlinkage long sys_saying(void){
4   printk("Veritas␣filia␣temporis.\n");
5
6   return 0;
7 }
```

   La función `printk()` permite mostrar un mensaje en pantalla en espacio de kernel (núcleo).

8. Cree el *Makefile* de `saying` con la siguiente línea:

   ```
   obj-y:=saying.o
   ```

   ¿Sabe qué es lo que está haciendo (uso de ':=')? Si no lo sabe, ¡averigüelo!

9. Acceda al directorio padre saliendo del directorio `practicaLab` y edite el *Makefile* del kernel. Añada en la línea `core-y += ...` el directorio `practicaLab/`. El objetivo es que, al compilar posteriormente el kernel, se considere el directorio creado y, por tanto, el archivo fuente de la nueva llamada al sistema.

10. Acceda al directorio `arch/x86/entry/syscalls` para realizar el registro de la nueva llamada al sistema (si su plataforma es 80x86) y edite el archivo `syscall_64.tbl` (si su arquitectura es de 64 bits) para añadir la nueva llamada al sistema a las ya existentes. Para ello, añada al final del archivo la siguiente fila:

    ```
    <numero_sc> 64 <directorio-nueva-sc> <nombre-funcion-sc>
    ```

    donde cada campo de la nueva fila debe sustituirlo por el valor adecuado correspondiente a la nueva llamada al sistema[3].

11. Acceda al directorio `include/linux` del kernel donde se encuentran, entre otros, el archivo de cabecera `syscall.h` con las declaraciones de las llamadas al sistema.

12. Edite `syscall.h` y añada el prototipo (declaración) de la nueva llamada al sistema creada, antes de `#endif`.

[3] 64 si su plataforma es de 64 bits.

13. Configure el kernel usando `make menuconfig`[4]. Finalmente, en el menú que aparezca, confirme que el sistema de archivos es *ext4* y guarde la configuración.

14. Compile el kernel con `make`. Esta tarea requiere mucho tiempo y más realizándola en una VM. Se recomienda utilizar todos los núcleos que posee el sistema para reducir al máximo el tiempo de ejecución de esta compilación (utilice para ello el parámetro `-j`*nproc* al ejecutar `make`[5]. Aún así, tenga paciencia porque esta tarea durará unas horas...

15. Instale el nuevo kernel en su VM con `make modules_install install`

4. Testear la nueva llamada al sistema

Para comprobar que la nueva llamada al sistema se ejecuta correctamente, realice los siguientes pasos:

1. Reiniciar la VM con `reboot`. Manteniendo pulsada la tecla *Shift* durante el arranque, se accede a GRUB. De este modo, podemos elegir la versión del kernel recién compilada e instalada.

2. Compruebe la versión del kernel con el comando adecuado.

3. Cree un directorio de prueba y, en él, un archivo `test_saying.c` con el siguiente contenido:

```
1  #include <stdio.h>
2  #include <linux/kernel.h>
3  #include <sys/syscall.h>
4  #include <unistd.h>
5
6  int main(){
7    long int result = syscall(<num_sc>);
8    printf("System␣call:␣sys_saying␣returned␣%ld␣\n", result);
9    return 0;
10 }
```

   Entienda este código. Por ejemplo, ¿qué debe indicar en sustitución de *<num_sc>* en la línea 7 del código? ¿Por qué?

4. Compile el programa generando el archivo ejecutable `test_saying`.

5. Ejecute `test_saying`.

6. Ejecute el comando `dmesg`. En primer lugar, averigüe para qué puede servir este comando y utilícelo adecuadamente para ver el mensaje mostrado por el kernel al ejecutar la nueva llamada al sistema.

[4] Para ello, se le requerirá, mediante sendos mensajes de error, tener instalado adicionalmente ciertos paquetes: *bison*, *flex*, *libelf-dev*, *libncurses5-dev*, *libssl-dev*, etc.

[5] *nproc* es el número de núcleos de su sistema.

5. Ejercicio

Para comprender cómo se implementa una llamada al sistema, añada una nueva llamada al sistema al núcleo contestando a las preguntas que se plantean.

1. Detalle los pasos que ha realizado para añadir la nueva llamada, indicando razonadamente y con claridad qué implican cada uno de ellos. Siga el procedimiento especificado en la sección 3. Es fundamental que sepa qué está haciendo en cada momento y lo que implica cada paso en el contexto general en el que trabaja.

2. Compruebe que funciona la llamada al sistema. Indique de forma razonada y con claridad los pasos seguidos. Use de apoyo el procedimiento de la sección 4.

3. Utilice el comando `strace` para comprobar que al ejecutar la prueba de la nueva llamada al sistema efectivamente se invoca a la misma.

4. Represente en un esquema general los archivos/estructuras que se van invocando/utilizando en la ejecución de la llamada al sistema recién añadida así como el espacio en el que se ejecutan (espacio de usuario o espacio de núcleo) y modo en el que se ejecutan (modo usuario o modo supervisor). Este apartado es esencial para visualizar cuáles han sido todos los pasos que ha realizado y la relación entre ellos, de forma global,l entendiendo así el mecanismo y la implementación completa de las llamadas al sistema en Linux. Sugerencia: los diagramas deben ser similares a los proporcionados en las diapositivas relacionadas con la ejecución de las llamadas al sistema en Linux del tema 2.

5. *Opcional*: La llamada al sistema que se pide añadir no tiene parámetros. Se plantea como ejercicio opcional añadir una llamada al sistema de dos parámetros. Proporcione un ejemplo de llamada al sistema de este tipo. ¿Qué variaría en el procedimiento de inclusión de dicha llamada respecto a la realizada anteriormente?.