Sistemas Operativos

Práctica 3 **Curso 2023/2024** 3º Curso

Las llamadas al sistema en Linux

Introducción

Las llamadas al sistema permiten a los programas solicitar servicios al sistema operativo de manera síncrona. Constituyen, por tanto, la forma habitual en que dichos programas interactuarán de forma explícita con el sistema operativo.

Esta práctica supondrá la primera aproximación del alumno a la utilización de servicios de Linux utilizando llamadas al sistema, así como a la modificación del núcleo para la creación de nuevas llamadas al sistema.

Las llamadas al sistema constituyen el principal mecanismo para que nuestros programas utilicen las funciones proporcionadas por el sistema operativo

En la Práctica 2 se ha aprendido a modificar y recompilar el núcleo de Linux, añadiendo funciones nuevas. Sin embargo, esta operación pierde bastante sentido si dichas funciones no se pueden ofrecer como nuevas llamadas al sistema para que puedan ser utilizadas por los programas de aplicación.

Objetivos

Esta práctica tiene tres objetivos fundamentales:

- Introducir el mecanismo de llamadas al sistema operativo de Linux.
- Añadir nuevas llamadas al sistema Linux.
- Aprender a intercambiar datos entre el núcleo y los procesos de usuario.

Entorno

Los ordenadores del laboratorio ya están preparados para realizar las prácticas.

Si el alumno desea realizar las prácticas en su ordenador deberá:

1. Tener instalado Linux (la distribución OpenSuse 12.3 PC de 32 bits)
2. Configurarlo siguiendo las indicaciones del documento que se puede encontrar en http://moovi.uvigo.es

Utilizando llamadas al sistema en Linux

Introducción

Cada sistema operativo define un conjunto propio de llamadas al sistema. El número de llamadas puede variar de algunas decenas a varios centenares en función del sistema operativo considerado. En el caso del sistema operativo Linux, este número está en torno a las 350, aunque algunas no se implementan en todas las arquitecturas y un conjunto importante de ellas son obsoletas, aunque se mantienen por motivos de compatibilidad con programas desarrollados para versiones anteriores, de tal manera que el número de llamadas de uso actual está en torno a las 170. En Linux, la atención a una llamada al sistema supone cuatro operaciones principales:

1. Pasar el procesador a modo *supervisor*.
2. Ejecutar la función del núcleo que implementa el servicio solicitado.
3. Pasar el procesador a modo *usuario*.
4. Devolver el resultado al proceso llamante.

Realizar una llamada directa por número

La librería C estándar de Linux proporciona una función que permite realizar cualquier llamada al sistema utilizando su identificador numérico. Dicha función se denomina `syscall` y tiene la siguiente definición:

```
int syscall(int number, ... )
```

Es decir, toma como primer parámetro el número de llamada al sistema a realizar y como parámetros adicionales aquéllos que se le pasarán a dicha llamada, devolviendo el valor entero que representa el resultado correspondiente.

Ejercicio 1

Escriba un programa en C que realice la llamada al sistema número `20`, que es una llamada que no toma ningún parámetro. El programa debe presentar el resultado en pantalla.

El valor que se obtiene como resultado de ejecutar esta llamada es el PID (*Process IDentifier*) del proceso que la realiza.

¿Por qué en cada ejecución se obtiene un valor de PID diferente?

Comprobar el resultado de una llamada

Una funcionalidad que nos será útil, sobre todo cuando probemos llamadas al sistema desarrolladas por nosotros, es la comprobación de si la ejecución de la llamada ha sido correcta o si, por el contrario, se ha producido algún error y, en este último caso, saber en qué consiste dicho error. Para ello, Linux define una función denominada `perror()` que, si la llamamos inmediatamente después de realizar una llamada la sistema, nos proporciona dicha información.

Ejercicio 2

Modifique el programa de la práctica anterior para que después de realizar la llamada al sistema presente en pantalla información sobre si la llamada se ha realizado correctamente o no.

Pruebe este programa con diferentes llamadas al sistema (20, 350, 3, …)

Pruebe el programa desarrollado con las llamadas al sistema 1 y 2. Intente entender el comportamiento del programa con estas llamadas.

Realizar una llamada directa utilizando etiquetas

El mecanismo que se acaba de utilizar para realizar una llamada al sistema presenta una dificultad obvia: es necesario conocer el número de cada una de las llamadas al sistema.

Por suerte, Linux define etiquetas con un nombre significativo para cada uno de los números de llamada al sistema. Estas etiquetas se encuentran en el fichero de cabeceras "`/usr/include/sys/syscall.h`"(que en laboratorio - Linux 32 bits para PC - es una referencia al fichero "`/usr/include/asm/unistd_32.h`)", y tienen la forma `__NR_xxxxx`. Así, si se examina dicho fichero, se puede ver que, por ejemplo, para la llamada número 1 se define la etiqueta `__NR_exit`.

Ejercicio 3

Modifique el programa del Ejercicio 2 para utilizar la etiqueta correspondiente en vez del valor numérico `20`

Realizar una llamada mediante funciones de librería

La función `syscall` utilizada es una función genérica que permite realizar de forma directa cualquier llamada al sistema disponible. Esta generalidad supone sin embargo algunas limitaciones. Así, por ejemplo, no es posible hacer ninguna comprobación sobre si los parámetros pasados a una llamada al sistema son correctos antes de realizarla, lo cual sería deseable. Esta comprobación no es posible porque parámetros del mismo tipo pueden admitir valores distintos en función del significado concreto que cada llamada le dé a ese parámetro. Así, dos llamadas pueden tomar un parámetro entero, pero para una ser válidos solo los valores positivos porque, por ejemplo, representan un PID y para la otra ser válidos también valores negativos, porque al parámetro se le da el significado de, por ejemplo, el número de posiciones a avanzar o retroceder la posición actual de una referencia al contenido de un fichero.

Para resolver este problema y, al mismo tiempo, proporcionar un entorno de llamadas al sistema más amigable para el programador, la librería estándar de C (denominada *libc*) define para cada una de las llamadas al sistema una función que realiza la correspondiente llamada, así como, si es necesario, un procesamiento previo de sus parámetros y/o posterior de sus resultados.

Así, para la llamada al sistema `20`, que nos permite obtener el PID del proceso que la realiza, la librería define la siguiente función:

```
pid_t getpid(void);
```

La utilización de estas funciones es la forma habitual de realizar llamadas al sistema por parte de los programadores, hasta el punto de que informalmente muchas veces se hace referencia a ellas como "llamadas al sistema". No obstante, hay que tener clara la diferencia: no son llamadas directas al sistema. Es más, la correspondencia uno a uno entre funciones y llamadas al sistema no es total, existiendo algunas llamadas para las que no existe función de librería y también funciones de librería que realizan varias llamadas o utilizan diferentes parámetros que la llamada correspondiente.

Ejercicio 4

Escriba una nueva versión del programa del Ejercicio 3 para realizar la llamada al sistema `20` utilizando la correspondiente función de librería.

Creación de nuevas llamadas al sistema

Creación de funciones que implementan las llamadas al sistema

A lo largo de esta práctica se van a crear varias llamadas al sistema para experimentar con diferentes escenarios que se puedan presentar.

Por lo tanto, el primer objetivo será disponer de un conjunto de funciones que posteriormente serán transformadas en llamadas al sistema. Como el objetivo de la práctica es conocer el procedimiento de transformar funciones en llamadas al sistema, y con el fin de eliminar posibles puntos de fallo, se van a crear un conjunto de funciones en el núcleo que realicen operaciones triviales, con el mero objetivo de que sirvan como base para alcanzar los objetivos planteados.

A lo largo de la práctica se va a trabajar sobre tres llamadas al sistema de complejidad creciente. Todas ellas, siguiendo el convenio de Linux, devolverán un valor entero (que se define como *long* para asegurar que tenga el mismo tamaño que un registro en la arquitectura local). Este valor entero tomará un valor negativo en caso de que se produzca un error. Las tres funciones que implementarán dichas llamadas al sistema tendrán las siguientes definiciones:

1. Una función sencilla que no tome parámetros y devuelva un entero:

   ```
   long sys_nprocesos(void)
   ```

   Esta función, en su primera versión devolverá un valor fijo (por ejemplo, 7)

2. Una función que tome como parámetros dos valores enteros y devuelva otro entero.

   ```
   long sys_nprocen(int p1, int p2)
   ```

   La primera versión de esta función devolverá el producto de los dos valores que se pasan como parámetro.

3. Una función que tome como parámetro una cadena de caracteres y devuelva un entero.

   ```
   long sys_mensaje(char *texto)
   ```

   Esta función, en su primera versión, imprimirá en la consola la cadena que se le pasa como parámetro y devolverá la longitud de dicha cadena, o -1 si se ha producido algún error. Además, devolverá en el parámetro *texto* la cadena "CORRECTO" si no se ha producido error y la cadena "ERROR" en caso contrario. Para desarrollar está función se puede utilizar como base la que se desarrolló en el Ejercicio 10 de la Práctica 2.

Ejercicio 5:

Programe en el núcleo de Linux las tres funciones descritas en este apartado. Cree ficheros nuevos para que contengan dichas funciones (por ejemplo, `laboratorio.c` en el directorio *kernel/*, y `laboratorio.h` en el directorio *include/linux/*).

Conversión de una función en llamada al sistema

Una vez se ha definido en el núcleo la función correspondiente a una nueva llamada al sistema, se debe definir el correspondiente "punto de entrada" al núcleo.

Como ya sabemos, Linux, como cualquier otro sistema operativo, gestiona una tabla en la que almacena referencias a las diferentes llamadas al sistema. Por tanto, añadiendo una referencia a nuestra función en dicha tabla ya se podrá

acceder a ella como nueva llamada al sistema utilizando el número que se corresponda con la posición de dicha referencia en la tabla (ya que el número de llamada se utiliza como índice en la tabla).

Dado que el proceso de realizar una llamada al sistema supone ejecutar una determinada instrucción del procesador (la correspondiente a provocar una excepción software), el código que implementa este proceso es dependiente de la arquitectura de la máquina que se esté utilizando y además está escrito en ensamblador (ya que no existe una sentencia en C para provocar una excepción). En concreto, en la versión del núcleo de Linux que estamos utilizando, este código, que incluye la tabla de llamadas al sistema, así como otra serie de modificaciones del código del núcleo que son necesarias, se generan automáticamente durante el proceso de compilación a partir de un fichero que define el conjunto de llamadas al sistema existente. Esto simplifica mucho definir una nueva llamada, ya que, con incluir su información en dicho fichero de definición, se realizarán todos los cambios necesarios en el código. Este fichero de definición de las llamadas al sistema es "*arch/x86/syscalls/syscall_32.tbl*". Por tanto, debemos modificar este fichero para introducir al final la definición de nuestras nuevas llamadas.

En teoría esto es todo lo que se debería hacer. Sin embargo, se debe realizar un cambio adicional en la declaración y en la definición de las funciones que se quieren utilizar como llamadas al sistema, que consiste en añadir la directiva de precompilación `asmlinkage` para indicar al compilador que dicha función se va invocar desde código escrito en ensamblador y que, por tanto, no se deberían utilizar optimizaciones de C, como, por ejemplo, pasar parámetros en registros del procesador en vez de en el *stack*.

Esta directiva se debe añadir al principio de la declaración y la definición de la función, y podría ser incluida, en el caso de la declaración, en el fichero "*include/linux/syscalls.h*". Por ejemplo:

```
asmlinkage long sys_nprocesos(void)
```

Ejercicio 6

Convierta las dos primeras funciones definidas en el Ejercicio 5 en llamadas al sistema.

Compruebe el correcto funcionamiento de dichas llamadas recompilando el núcleo y utilizando el programa realizado en el Ejercicio 2 para invocar dichas llamadas.

Siguiendo los convenios de Linux

En las últimas versiones del núcleo de Linux, y en aras de aumentar el porcentaje del código del núcleo que es directamente portable a cualquier arquitectura, se han introducido una serie de macros que es conveniente utilizar. Así:

1. Para definir las llamadas al sistema se deberían utilizar las macros *SYSCALL_DEFINE0*, *SYSCALL_DEFINE1*, *SYSCALL_DEFINE2*, …, en las que el número que aparece al final indica el número de parámetros de la llamada.

   Por ejemplo, se debería sustituir la definición

   ```
   asmlinkage long sys_nprocesos(void)
   ```

   por

   ```
   SYSCALL_DEFINE0(nprocesos)
   ```

   Este cambio se debe hacer sólo para la definición de la función, no para la declaración.

   Para ver cómo se utilizan estas macros en funciones con parámetros, observe algunas definiciones, por ejemplo, en el fichero *kernel/sys.c*

2. Se debe añadir al final del fichero *kernel/sys_ni.c* una entrada para cada una de las nuevas llamadas igual a las ya existentes para las demás llamadas.

Además, existen algunos convenios que también se deberían seguir. Así:

1. La función que implementa la funcionalidad de una determinada llamada en el núcleo debería denominarse como la llamada, con el prefijo "*sys_*".
2. La etiqueta que se defina en el fichero *unistd.h* correspondiente debe tener el mismo nombre que la llamada al sistema, con el prefijo "*__NR_*"[1].

[1] Esto ya lo hace automáticamente el proceso de compilación del núcleo a partir de la tabla de definición de llamadas al sistema, generando una nueva versión del fichero en el directorio *"arch/x86/include/generated/uapi/asm"* de las fuentes del núcleo. Si queremos que los programas de usuario utilicen estas etiquetas deberemos copiar esta nueva versión a su ubicación habitual (*"/usr/include/asm"*)

A partir de ahora, cada vez que se defina una nueva llamada, se deberá hacer siguiendo todas las indicaciones anteriores.

Ejercicio 7

Modifique las llamadas al sistema del Ejercicio 6 para que sigan los convenios presentados en este apartado.

Compruebe que se pueden utilizar las etiquetas para realizar las nuevas llamadas mediante el programa desarrollado en el Ejercicio 3.

¿Por qué realizamos la modificación indicada para el fichero "*kernel/sys_ni.c*"?

Creación de una llamada al sistema con argumentos pasados por referencia

El paso de parámetros por referencia (paso de variables de tipo puntero) a una llamada al sistema presenta problemas motivados por el cambio de modo de ejecución (de modo usuario a modo supervisor). En concreto, el núcleo y cada uno de los procesos de usuario utilizan espacios de direccionamiento distintos y además el seguimiento directo de punteros en modo supervisor puede dar lugar a problemas de seguridad.

Para evitar estos problemas, el sistema operativo Linux define una serie de funciones internas del núcleo que permiten copiar datos entre el espacio de direccionamiento del proceso de usuario que realiza una llamada al sistema y el espacio de direccionamiento del núcleo.

Siempre se deben utilizar estas funciones en la implementación de nuevas llamadas al sistema, tanto para leer datos de entrada como para devolver resultados a través de punteros.

Las funciones que proporciona el núcleo de Linux son:

Función	Descripción
`access_ok`	Comprueba la validez del puntero en el espacio de memoria de usuario.
`get_user`	Copia una variable simple desde el espacio de usuario.
`put_user`	Copia una variable simple hacia el espacio de usuario.
`clear_user`	Rellena con ceros una zona de memoria en el espacio de usuario.
`copy_to_user`	Copia un bloque de datos desde el espacio del núcleo al espacio de usuario.
`copy_from_user`	Copia un bloque de datos desde el espacio del usuario al espacio del núcleo.
`strnlen_user`	Obtiene el tamaño de un *string* en el espacio de usuario.
`strncpy_from_user`	Copia un *string* desde el espacio del usuario al espacio del núcleo.

Ejercicio 8

Utilice estas funciones para crear una nueva llamada al sistema a partir de la tercera función desarrollada en el Ejercicio 5.