Las señales en Linux

Horacio Goetendía Bonilla

13 de Julio de 2003

Índice

1. Conceptos

1.1. Señal

Una señal es una interrupción que llega a un proceso. El sistema operativo usa señales para reportar situaciones excepcionales en la ejecución de un programa.

Si anticipas un evento que causa una señal entonces puedes definir una función *handler* o manejador de señales y decirle al Sistema Operativo que corra una función en específico cuando la señal llegue.

Otra característica es que un proceso puede mandar una señal a otro proceso por ejemplo para que un padre aborte a un hijo o para que dos procesos se comuniquen y "sincronicen".

Una señal reporta la ocurrencia de un suceso excepcional ejemplo:

- División entre cero.
- Un usuario manda con el teclado la terminación de un programa (Ctrl+C).
- La terminación de un proceso hijo.
- Expiración de un timer de alarma.
- Una llamada a *kill*.
- Hacer una operación de Entrada/Salida que no puede ser realizada (leer un pipe que no es de escritura) [1]

1.2. Generación de Señales

Las señales pueden ser generadas sincrónicas o asíncronamente. Los eventos que generan una señal caen dentro de 3 categorías.

Errores: El programa hizo algo invalido y no puede continuar su ejecución.

Eventos externos: Esto tiene que ver con Entradas/Salidas o con otros procesos (llegada de una entrada, expiración de un timer, terminación de un hijo.)

Llamadas explicitas(request): El uso de funciones de librería tal como el *kill* para generar una señal.

[1] Las señales son limitadas pero muy usadas para la comunicación interprocesos.

1.3. ¿Como son enviadas las señales?

Cuando una señal es generada esta se dice que es una señal pendiente o que llega a ser pendiente (solo es pendiente por un pequeño lapso de tiempo). Sin embargo si una señal pendiente es bloqueada esta puede quedar en estado pendiente indefinidamente.

Ciertas señales tales como las SIGKILL y SIGSTOP no se pueden interceptar o ignorar y realizan la acción predeterminada asociada a la señal.

Para toda señal (excepto SIGKILL y SIGSTOP) se puede:

- Ignorar la señal.
- Especificar una función específica[2].
- Aceptar la acción por defecto que trae con sigo la señal.

El programa especifica la acción *handler* que se desea realizar usando funciones tal como *signal* o *signalaction*.

Muchas veces decimos que una señal es atrapada *(catch)* por un *handler*. Mientras el *handler* esta corriendo en el momento de la terminación.

1.4. Señales de control de Tareas

Como señales de control de tareas tenemos las siguientes:

SIGCHLD La señal es enviada a un proceso padre cuando un hijo termina o pare. Equivalente(17) .

- Acción por defecto: Ignorarla.
- Handler: Definido por el programador.

SIGCONT Hace que un proceso continúe. Equivalente(18)

SIGSTOP Para un proceso esta señal no puede ser manejada o ignorada. Equivalente(19)

SIGTSTP Es un SIGSTOP interactivo. Esta puede ser manejada o ignorada (Ejem CRTL+C o +Z). Equivalente(20)

SIGTTIN Cuando un proceso en segundo plano trata de leer un terminal. La acción por defecto es parar el proceso. Equivalente(21).

SIGTTOUT Cuando un proceso en segundo plano trata de escribir en un terminal. Equivalente(22)

[2]A esta función se le denomina *handler*

Existen al rededor de 32 señales en Linux. Cada señal esta asociada a un entero numérico equivalente.

Cuando un proceso esta parado otras señales no pueden ser entregadas hasta que este continúe excepto SIGKILL o SIGCONT, todas las señales que fueron tratadas de ser entregadas pasan a ser pendientes hasta que el proceso continúe.

2. Manejo básico de señales

Esta llamada establece el comportamiento para la señal:

`signal (int signum, sighandler_t accion)`
`signum: Es la señal a la cual se le va asociar el comportamiento`
`sighandler_t: Es el tipo de las funciones manejadoras o` ***handlers***

La acción o handler es determinada de la siguiente manera:

Si `signal (int signum, sighandler_t accion)` visto anteriormente accion es seteada a SIG_DFL se esta especificando la acción por defecto asociada a la señal `signum`. Si en vez de SIG_DFL esta SIG_IGN se especifica que la señal debe de ser ignorada.

Cuando uno desea que una señal no sea entregada en cierta parte del programa, se debe bloquear no ignorar.

signal cuando falla retorna un SIG_ERR, Si tiene éxito *signal* retorna la acción que previamente tenía efecto para la señal tratada, la cual puede ser guardada y retornada a la normalidad nuevamente, después del llamado a signal.

2.1. Haciendo que el handler se comporte como quieres

La función *sigaction* tiene el mismo efecto que *signal* pero *sigaction* ofrece más control y claro es un poco más complicado de manejar.

sigaction ofrece banderas específicas adicionales para el control de en que momento la señal debería ser generada y de manera será invocado el *handler*.

2.1.1. La estructura *sigaction*

La estructura *sigaction* es usada en la función *sigaction*, y esta esta determinada como sigue:

```
struct   sigaction
         {
         sighandler_t   sa_handler   La acción.
         sigset_t       sa_mask      Especifica un set de señales a ser
                                     bloqueadas mientras el handler corre.
         int            sa_flags     Especifica las banderas que afectan
                                     el comportamiento de la señal.
         }
```

2.1.2. La función *sigaction*

```
sigaction (int signum, const struct sigaction *action, struct sigaction *oldaction)
```

signum: Es la señal asociada a la acción.

action: Si action es NULL la acción asociada a la señal no cambia. Esto sirve para verificar como la señal estaba siendo manipulada.

oldaction: Si oldaction es NULL simplemente se suprime el retorno de información acerca de la acción anterior.

sigaction retorna 0 si todo va bien y -1 si hubo algún error.

Referencias

[1] Sandra Loosemore : *The GNU C Library Reference Manual*
Free Software Foundation, 1999

[2] Mark Mitchell, Jeffrey Oldham, Alex Samuel : *Advanced Linux Programming*
New Riders, 2001

[3] Kay A. Robbins, Steven Robbins : *Unix Programación Práctica*
Prentice Hall, Primera Edición

[4] Kurt Wall : *Programación en Linux con ejemplos*
Prentice Hall, 2000