MÓDULO II. Uso de los Servicios del SO Linux mediante la API

Sesión 5. Llamadas al sistema para gestión y control de señales

2025/08/20

Índice

1. Objetivos principales

En esta sesión trabajaremos con las llamadas al sistema relacionadas con la **gestión y el control de señales**. El control de señales en Linux incluye las llamadas al sistema necesarias para cambiar el comportamiento de un proceso cuando recibe una determinada señal, examinar y cambiar la máscara de señales y el conjunto de señales bloqueadas, y suspender un proceso, así como comunicar procesos.

- Conocer las llamadas al sistema para el control de señales.
- Conocer las funciones y las estructuras de datos que me permiten trabajar con señales.
- Aprender a utilizar las señales como mecanismo de comunicación entre procesos.

2. Señales

Las señales constituyen un mecanismo básico de **sincronización** que utiliza el núcleo de Linux para indicar a los procesos la ocurrencia de determinados eventos síncronos/asíncronos con su ejecución. Aparte del uso de señales por parte del núcleo, los procesos pueden enviarse señales para la notificación de cierto evento (la señal es generada cuando ocurre este evento) y, lo que es más importante, pueden determinar qué acción realizarán como respuesta a la recepción de una señal determinada.

Un **manejador de señal** es una función definida en el programa que se invoca cuando se entrega una señal al proceso. La invocación del manejador de la señal puede interrumpir el flujo de control del proceso en cualquier instante. Cuando se entrega la señal, el *kernel* invoca al manejador, y cuando el manejador retorna, la ejecución del proceso sigue por donde fue interrumpida, como muestra la figura 1.

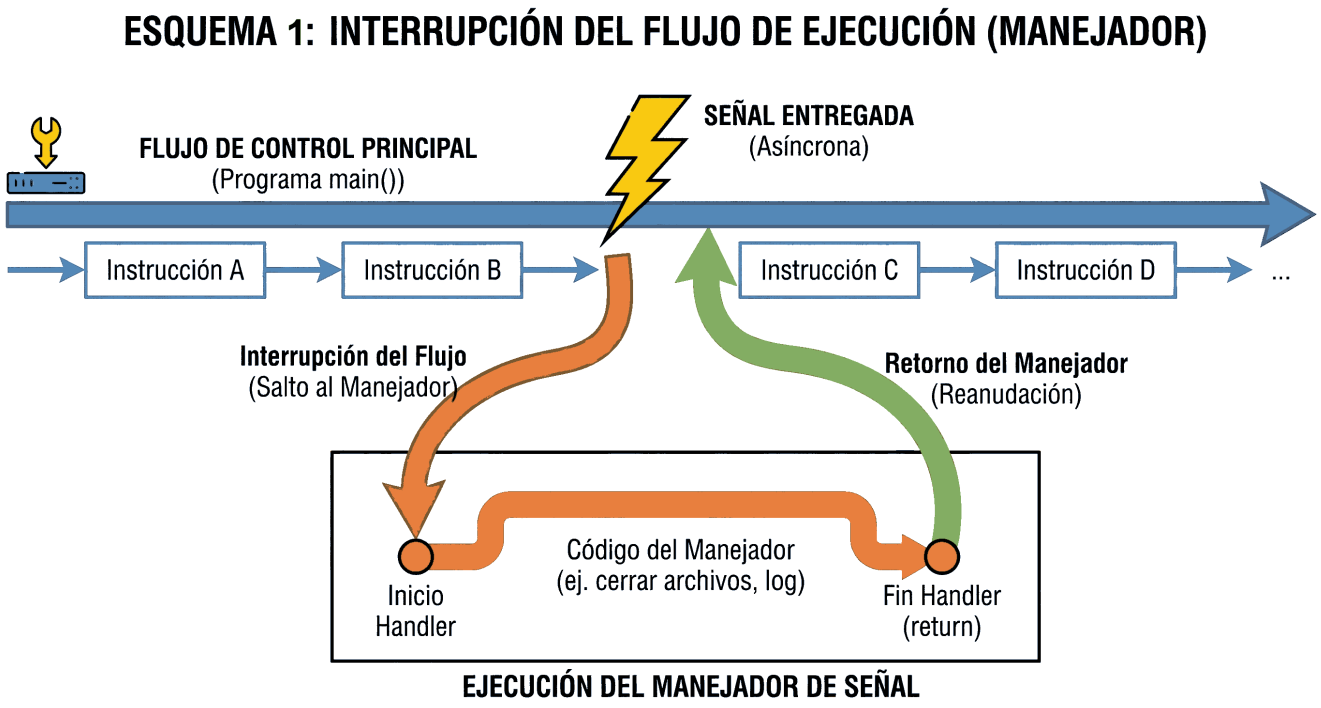

Figura 1: **Figura 1.** Ejecución del manejador tras la entrega de la señal.

Se dice que una señal es **depositada** cuando el proceso inicia una acción en base a ella, y se dice que una señal está **pendiente** si ha sido generada pero todavía no ha sido depositada. Además un proceso puede bloquear la recepción de una o varias señales a la vez.

Las señales bloqueadas de un proceso se almacenan en un conjunto de señales llamado **máscara de bloqueo de señales**. No se debe confundir una señal bloqueada con una señal ignorada, ya que una señal ignorada es desechada por el proceso, mientras que una señal bloqueada permanece pendiente y será depositada cuando el proceso la desenmascare (la desbloquee). Si una señal es recibida varias veces mientras está bloqueada, se maneja como si se hubiese recibido una sola vez.

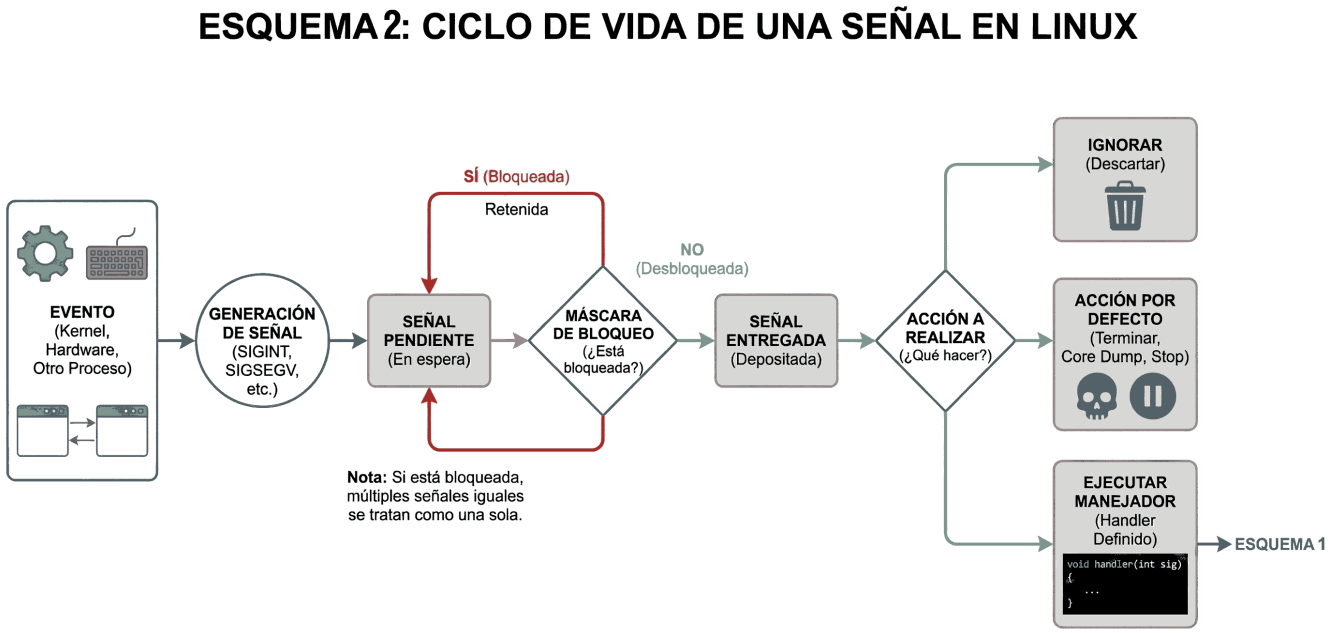

Figura 2: **Figura 2.** Esquema de manejo de señales por el SO.

La lista de señales y su tratamiento por defecto se puede consultar con `man 7 signal` (o en `signal.h`). En la lista siguiente se muestran las señales posibles en POSIX.1. Cada señal posee un nombre que comienza por `SIG`, mientras que el resto de los caracteres se relacionan con el tipo de evento que representa. Realmente, cada señal lleva asociado un número entero positivo, que es el que se entrega al proceso cuando éste recibe la señal. Se puede usar indistintamente el número o la constante que representa a la señal.

Lista de señales en **POSIX.1**. En negrita el **símbolo**, entre paréntesis y cursiva la *acción*, y a la derecha su significado:

- **SIGHUP** (*Term*): Desconexión del terminal (referencia a la función `termio(7)` del man). También se utiliza para reanudar los demonios `init`, `httpd` e `inetd`. Esta señal la envía un proceso padre a un proceso hijo cuando el padre finaliza.
- **SIGINT** (*Term*): Interrupción procedente del teclado (`<Ctrl+C>`).
- **SIGQUIT** (*Core*): Terminación procedente del teclado.

- **SIGILL** (*Core*): Excepción producida por la ejecución de una instrucción ilegal.
- **SIGABRT** (*Core*): Señal de aborto procedente de la llamada al sistema `abort(3)`.
- **SIGFPE** (*Core*): Excepción de coma flotante.
- **SIGKILL** (*Term*): Señal para terminar un proceso (no se puede ignorar ni manejar). Se corresponde con el número 9. Es la señal que se envía con `kill -9` para "matar" un proceso de forma forzosa.
- **SIGSEGV** (*Core*): Referencia inválida a memoria.
- **SIGPIPE** (*Term*): Tubería rota: escritura sin lectores.
- **SIGALRM** (*Term*): Señal de alarma procedente de la llamada al sistema `alarm(2)`.
- **SIGTERM** (*Term*): Señal de terminación.
- **SIGUSR1** (*Term*): Señal definida por el usuario (1).
- **SIGUSR2** (*Term*): Señal definida por el usuario (2).
- **SIGCHLD** (*Ign*): Proceso hijo terminado o parado.
- **SIGCONT** (*Cont*): Reanudar el proceso si estaba parado.
- **SIGSTOP** (*Stop*): Parar proceso (no se puede ignorar ni manejar). Equivalente a cuando se pulsa `<Ctrl+Z>`.
- **SIGTSTP** (*Stop*): Parar la escritura en la `tty`.
- **SIGTTIN** (*Stop*): Entrada de la `tty` para un proceso de fondo.
- **SIGTTOU** (*Stop*): Salida a la `tty` para un proceso de fondo.

Las entradas entre paréntesis (acción) de la lista anterior especifican la acción por defecto para la señal usando la siguiente nomenclatura:

- *Term*: La acción por defecto es terminar el proceso.
- *Ign*: La acción por defecto es ignorar la señal.
- *Core*: La acción por defecto es terminar el proceso y realizar un volcado de memoria.
- *Stop*: La acción por defecto es detener el proceso.
- *Cont*: La acción por defecto es que el proceso continúe su ejecución si está parado.

Puedes ver la lista de señales al completo usando la orden `kill -l`.

Las llamadas al sistema que podemos utilizar en Linux para trabajar con señales son principalmente:

- `kill`: se utiliza para enviar una señal a un proceso o conjunto de procesos. Valor de retorno 0 en caso de éxito y -1 en caso de error.
- `sigaction`: permite establecer la acción que realizará un proceso como respuesta a la recepción de una señal. Las únicas señales que no pueden cambiar su acción por defecto son: `SIGKILL` y `SIGSTOP`.
- `sigprocmask`: se emplea para cambiar la lista de señales bloqueadas actualmente.
- `sigpending`: permite el examen de señales pendientes (las que se han producido mientras estaban bloqueadas).
- `sigsuspend`: reemplaza temporalmente la máscara de señal para el proceso con la dada por el argumento `mask` y luego suspende el proceso hasta que se recibe una señal.

Sinopsis

```
#include <signal.h>

int kill(pid_t pid, int sig);
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
int sigpending(sigset_t *set);
int sigsuspend(const sigset_t *mask);
```

2.1 La llamada `kill`

La llamada `kill` se puede utilizar para enviar cualquier señal a un proceso o grupo de procesos.

Sinopsis

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

Argumentos

- Si `pid` es positivo, entonces se envía la señal `sig` al proceso con identificador de proceso igual a `pid`. Si `pid` es igual a -1, entonces `sig` se envía a cada proceso para el cual el proceso que realiza la llamada tiene permiso de enviar señales, excepto al proceso 1 (`init`). Si `pid` es menor que -1, entonces `sig` se envía a cada proceso en el grupo de procesos cuyo identificador es **-pid**.
- Si `pid` es `0`, entonces `sig` se envía a cada proceso en el grupo de procesos del proceso actual.
- Si `pid` es igual a `–1`, entonces se envía la señal `sig` a cada proceso, excepto al primero, desde los números más altos en la tabla de procesos hasta los más bajos.
- Si `pid` es menor que `–1`, entonces se envía `sig` a cada proceso en el grupo de procesos `–pid`.
- Si `sig` es `0`, entonces no se envía ninguna señal, pero sí se realiza la comprobación de errores.

2.2 La llamada `sigaction`

La llamada al sistema `sigaction` se emplea para cambiar la acción tomada por un proceso cuando recibe una determinada señal.

Sinopsis

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

Argumentos

El significado de los parámetros de la llamada es el siguiente:

- `signum` especifica la señal y puede ser cualquier señal válida salvo `SIGKILL` o `SIGSTOP`.
- Si `act` no es `NULL`, la nueva acción para la señal `signum` se instala desde `act`.
- Si `oldact` no es `NULL`, la acción anterior se guarda en `oldact`.

Valor de retorno

`0` en caso de éxito y `–1` en caso de error.

Estructuras de datos

La estructura `sigaction` se define como:

```
struct sigaction {
    void (*sa_handler)(int);
```

```
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

- `sa_handler` especifica la acción que se va a asociar con la señal `signum` pudiendo ser:
  - `SIG_DFL` para la acción predeterminada,
  - `SIG_IGN` para ignorar la señal, o
  - un puntero a una función manejadora para la señal.
- `sa_mask` permite establecer una máscara de señales que deberían bloquearse durante la ejecución del manejador de la señal. Además, la señal que lance el manejador será bloqueada, a menos que se activen las opciones `SA_NODEFER` o `SA_NOMASK`. Para asignar valores a `sa_mask`, se usan las siguientes funciones:
  - `int sigemptyset(sigset_t *set);` inicializa a vacío un conjunto de señales (devuelve `0` si tiene éxito y `-1` en caso contrario).
  - `int sigfillset(sigset_t *set);` inicializa un conjunto con todas las señales (devuelve `0` si tiene éxito y `-1` en caso contrario).
  - `int sigismember(const sigset_t *set, int senyal);` determina si una señal `senyal` pertenece a un conjunto de señales `set` (devuelve `1` si la señal se encuentra dentro del conjunto, y `0` en caso contrario).
  - `int sigaddset(sigset_t *set, int signo);` añade una señal a un conjunto de señales `set` previamente inicializado (devuelve `0` si tiene éxito y `-1` en caso contrario).
  - `int sigdelset(sigset_t *set, int signo);` elimina una señal `signo` de un conjunto de señales `set` (devuelve `0` si tiene éxito y `-1` en caso contrario).
- `sa_flags` especifica un conjunto de opciones que modifican el comportamiento del proceso de manejo de señales. Se forma por la aplicación del operador de bits OR a cero o más de las siguientes constantes:
  - `SA_NOCLDSTOP`: Si `signum` es `SIGCHLD`, indica al núcleo que el proceso no desea recibir notificación cuando los procesos hijos se paren (esto es, cuando los procesos hijos reciban una de las señales: `SIGTSTP`, `SIGTTIN` o `SIGTTOU`).
  - `SA_ONESHOT` o `SA_RESETHAND`: Indica al núcleo que restaure la acción para la señal al estado predeterminado una vez que el manejador de señal haya sido llamado.
  - `SA_RESTART`: Proporciona un comportamiento compatible con la semántica de señales de BSD haciendo que ciertas llamadas al sistema reinicien su ejecución cuando son interrumpidas por la recepción de una señal.
  - `SA_NOMASK` o `SA_NODEFER`: Se pide al núcleo que no impida la recepción de la señal desde el propio manejador de la señal.
  - `SA_SIGINFO`: El manejador de señal toma 3 argumentos, no uno. En este caso, se debe configurar `sa_sigaction` en lugar de `sa_handler`. El parámetro `siginfo_t` para `sa_sigaction` es una estructura con los siguientes elementos:

```
siginfo_t {
    int si_signo; /* Número de señal */
    int si_errno; /* Un valor errno */
    int si_code; /* Código de señal */
    pid_t si_pid; /* ID del proceso emisor */
    uid_t si_uid; /* ID del usuario real del proceso emisor */
    int si_status; /* Valor de salida o señal */
    clock_t si_utime; /* Tiempo de usuario consumido */
    clock_t si_stime; /* Tiempo de sistema consumido */
```

```
    sigval_t si_value; /* Valor de señal */
    int si_int; /* señal POSIX.1b */
    void *si_ptr; /* señal POSIX.1b */
    void *si_addr; /* Dirección de memoria que ha producido el fallo */
    int si_band; /* Evento de conjunto */
    int si_fd; /* Descriptor de fichero */
}
```

Los posibles valores para cualquier señal se pueden consultar con `man sigaction`.

Nota: El elemento `sa_restorer` está obsoleto y no debería utilizarse. POSIX no especifica un elemento `sa_restorer`.

Los siguientes ejemplos ilustran el uso de la llamada al sistema `sigaction` para establecer un manejador para la señal `SIGINT` que se genera cuando se pulsa `<CTRL+C>`.

```c
/* tarea9.c */
#include <stdio.h>
#include <signal.h>

int main(){
   struct sigaction sa;
   sa.sa_handler = SIG_IGN; /* ignora la señal */
   sigemptyset(&sa.sa_mask);

   /* Reiniciar las funciones que hayan sido interrumpidas por un manejador */
   sa.sa_flags = SA_RESTART;

   if (sigaction(SIGINT, &sa, NULL) == -1){
      printf("error en el manejador");}
   while(1);
}
```

```c
/* tarea10.c */
#include <stdio.h>
#include <signal.h>

static int s_recibida=0;
static void handler (int signum){
   printf("\n Nueva acción del manejador \n");
   s_recibida++;}


int main()
{
   struct sigaction sa;
   sa.sa_handler = handler; /* establece el manejador a handler */
   sigemptyset(&sa.sa_mask);

   /* Reiniciar las funciones que hayan sido interrumpidas por un manejador */
   sa.sa_flags = SA_RESTART;
```

```
   if (sigaction(SIGINT, &sa, NULL) == -1){
      printf("error en el manejador");}
   while(s_recibida<3);
}
```

📖 **Actividad 5.1. Trabajo con las llamadas al sistema `sigaction` y `kill`**

A continuación se muestra el código fuente de dos programas. El programa `envioSignal` permite el envío de una señal a un proceso identificado por medio de su PID. El programa `reciboSignal` se ejecuta en *background* y permite la recepción de señales.

📝 **Ejercicio 1.** Compila y ejecuta los siguientes programas y trata de entender su funcionamiento.

```
/*
  envioSignal.c
  Trabajo con llamadas al sistema del Subsistema de Procesos conforme a POSIX 2.10
  Utilización de la llamada kill para enviar una señal:
  0: SIGTERM
  1: SIGUSR1
  2: SIGUSR2
  a un proceso cuyo identificador de proceso es PID.
  SINTAXIS: envioSignal [012] <PID>
*/

#include <sys/types.h> /* POSIX Standard: 2.6 Primitive System Data Types */
/* <sys/types.h> */
#include<limits.h> /* Incluye <bits/posix1_lim.h> POSIX Standard: 2.9.2
                      Minimum Values Added to <limits.h> y <bits/posix2_lim.h> */
#include <unistd.h> /* POSIX Standard: 2.10 Symbolic Constants <unistd.h> */
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
int main(int argc, char *argv[])
{
   long int pid;
   int signal;
   if(argc<3) {
      printf("\nSintaxis de ejecución: envioSignal [012] <PID>\n\n");
      exit(-1);
   }
   pid= strtol(argv[2],NULL,10);
   if(pid == LONG_MIN || pid == LONG_MAX)
   {
      if(pid == LONG_MIN)
         printf("\nError por desbordamiento inferior LONG_MIN %d",pid);
      else
```

```
            printf("\nError por desbordamiento superior LONG_MAX %d",pid);
         perror("\nError en strtol");
         exit(-1);
      }
      signal=atoi(argv[1]);
      switch(signal) {
         case 0: /* SIGTERM */
            kill(pid,SIGTERM); break;
         case 1: /* SIGUSR1 */
            kill(pid,SIGUSR1); break;
         case 2: /* SIGUSR2 */
            kill(pid,SIGUSR2); break;
         default : /* not in [012] */
            printf("\n No puedo enviar ese tipo de señal");
      }
}
/*
 * reciboSignal.c
 * Trabajo con llamadas al sistema del Subsistema de Procesos conforme a POSIX 2.10
 * Utilización de la llamada sigaction para cambiar el comportamiento del proceso
 * frente a la recepción de una señal.
 */

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <errno.h>

/*
 * Manejador de señal para SIGUSR1 y SIGUSR2.
 * Esta función se llama cuando el proceso recibe una de estas señales.
 */
static void sig_USR_hdlr(int sigNum)
{

   if(sigNum == SIGUSR1)
      printf("\nRecibida la señal SIGUSR1\n\n");
   else if(sigNum == SIGUSR2)
      printf("\nRecibida la señal SIGUSR2\n\n");
}

int main(int argc, char *argv[])
{
   /*
    * Declaración de la estructura sigaction.
    * Esta estructura se usa para definir la nueva acción de la señal.
    */
```

```c
struct sigaction sig_USR_nact;

/*
 * Deshabilita el buffering de la salida estándar para que
 * los mensajes se impriman inmediatamente.
 */
if(setvbuf(stdout,NULL,_IONBF,0))
{
   perror("\nError en setvbuf");
}

/*
 * Inicializa la estructura sig_USR_nact para especificar la nueva acción
 * para la señal. Asigna la función 'sig_USR_hdlr' como manejador de señal.
 */
sig_USR_nact.sa_handler= sig_USR_hdlr;


/*
 * Inicia el conjunto de señales bloqueadas 'sa_mask' al conjunto vacío.
 * Esto significa que no se bloquearán señales adicionales mientras se ejecuta
 * el manejador.
 */
sigemptyset (&sig_USR_nact.sa_mask);

/*
 * Establece las banderas de la estructura. sa_flags en 0 significa que no se
 * usan opciones especiales.
 */
sig_USR_nact.sa_flags = 0;

/*
 * Establece el manejador de señal para SIGUSR1 usando la llamada al sistema
 * sigaction. La función 'sigaction' cambia la acción del proceso al recibir
 * la señal SIGUSR1.
 */
if( sigaction(SIGUSR1,&sig_USR_nact,NULL) <0)
{
   perror("\nError al intentar establecer el manejador de señal para SIGUSR1");
   exit(-1);
}

/*
 * Establece el manejador de señal para SIGUSR2 usando la llamada al sistema
 * sigaction.
 */
if( sigaction(SIGUSR2,&sig_USR_nact,NULL) <0)
{
   perror("\nError al intentar establecer el manejador de señal para SIGUSR2");
```

```
        exit(-1);
    }

    /*
     * Bucle infinito para mantener el proceso en ejecución y esperando señales.
     */
    for(;;)
    {
    }
}
```

📝 **Ejercicio 2.** Escribe un programa en C llamado `contador`, tal que cada vez que reciba una señal que se pueda manejar, muestre por pantalla la señal y el número de veces que se ha recibido ese tipo de señal, y un mensaje inicial indicando las señales que no puede manejar. En el cuadro siguiente se muestra un ejemplo de ejecución del programa.

```
kawtar@kawtar-VirtualBox:~$ ./contador &
[2] 1899
kawtar@kawtar-VirtualBox:~$
No puedo manejar la señal 9
No puedo manejar la señal 19
Esperando el envío de señales...
kill -SIGINT 1899
kawtar@kawtar-VirtualBox:~$ La señal 2 se ha recibido 1 veces
kill -SIGINT 1899
La señal 2 se ha recibido 2 veces
kill -15 1899
kawtar@kawtar-VirtualBox:~$ La señal 15 se ha recibido 1 veces
kill -111 1899
bash: kill: 111: especificación de señal inválida
kawtar@kawtar-VirtualBox:~$ kill -15 1899 # no puede capturar la señal 15
[2]+ Detenido                 ./contador
kawtar@kawtar-VirtualBox:~$ kill -cont 1899
La señal 18 se ha recibido 1 veces
kawtar@kawtar-VirtualBox:~$ kill -KILL 1899
[2]+  Terminado (killed)    ./contador
```

2.3 La llamada `sigprocmask`

La llamada `sigprocmask` se emplea para examinar y cambiar la máscara de señales.

Sinopsis

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

Argumentos

- El argumento how indica el tipo de cambio. Los valores que puede tomar son los siguientes:

  - **SIG_BLOCK**: El conjunto de señales bloqueadas es la unión del conjunto actual y el argumento `set`.
  - **SIG_UNBLOCK**: Las señales que hay en `set` se eliminan del conjunto actual de señales bloqueadas. Es posible intentar el desbloqueo de una señal que no está bloqueada.
  - **SIG_SETMASK**: El conjunto de señales bloqueadas se pone según el argumento `set`.
- `set` representa el puntero al nuevo conjunto de señales enmascaradas. Si `set` es diferente de `NULL`, apunta a un conjunto de señales, en caso contrario `sigprocmask` se utiliza para consulta.
- `oldset` representa el conjunto anterior de señales enmascaradas. Si `oldset` no es `NULL`, el valor anterior de la máscara de señal se guarda en `oldset`. En caso contrario no se retorna la máscara anterior.

Valor de retorno

`0` en caso de éxito y `–1` en caso de error.

2.4 La llamada `sigpending`

La llamada `sigpending` permite examinar el conjunto de señales bloqueadas y/o pendientes de entrega. La máscara de señal de las señales pendientes se guarda en `set`.

Sinopsis

```
int sigpending(sigset_t *set);
```

Argumento

- `set` representa un puntero al conjunto de señales pendientes.

Valor de retorno

- `0` en caso de éxito y `–1` en caso de error.

2.5 La llamada `sigsuspend`

La llamada `sigsuspend` reemplaza temporalmente la máscara de señal para el proceso con la dada por el argumento `mask` y luego suspende el proceso hasta que se recibe una señal.

Sinopsis

```
int sigsuspend(const sigset_t *mask);
```

Argumento

- `mask` representa el puntero al nuevo conjunto de señales enmascaradas.

Valor de retorno

- `–1` si `sigsuspend` es interrumpida por una señal capturada (no está definida la terminación correcta).

Ejemplo de uso

En el siguiente ejemplo se suspende la ejecución del proceso actual hasta que reciba una señal distinta de `SIGUSR1`.

```c
/* tarea11.c */

#include <stdio.h>
#include <signal.h>

int main(){
   sigset_t new_mask;

   /* inicializar la nueva mascara de señales */
   sigemptyset(&new_mask);

   sigaddset(&new_mask, SIGUSR1);

   /*esperar a cualquier señal excepto SIGUSR1 */
   sigsuspend(&new_mask);
}
```

Notas finales

A continuación se presentan algunas notas finales sobre la gestión de señales en Linux y su conformidad con los estándares POSIX:

- **Bloqueo de señales**: No es posible bloquear las señales `SIGKILL` ni `SIGSTOP` con la llamada `sigprocmask`. Cualquier intento de hacerlo será ignorado por el núcleo.
- **Señales con comportamiento indefinido**: Según el estándar POSIX, el comportamiento de un proceso es indefinido si ignora las señales `SIGFPE`, `SIGILL` o `SIGSEGV`, a menos que hayan sido generadas por las llamadas `kill` o `raise`. Por ejemplo, una división por cero puede generar una señal `SIGFPE` en algunas arquitecturas. Ignorar esta señal podría provocar un bucle infinito.
- **Uso de `sigaction`**:
  - Es posible llamar a `sigaction` con su segundo argumento (`act`) como `NULL` para obtener el manejador de señal actual.
  - También se puede usar para verificar si una señal es válida en el sistema, llamándola con los argumentos `act` y `oldact` como `NULL`.
- **Compatibilidad y estándares**:
  - **`SIGCHLD`**: POSIX (B.3.3.1.3) anula el comportamiento de ignorar la señal `SIGCHLD` (`SIG_IGN`). Los programas que asignan `SIG_IGN` a `SIGCHLD` pueden fallar en Linux debido a las diferencias entre los comportamientos de BSD y SYSV.
  - **`sa_flags`**: La especificación POSIX solo define `SA_NOCLDSTOP`. El uso de otros valores en `sa_flags` no es portable entre distintos sistemas.
  - **Opciones `SA_RESETHAND` y `SA_NODEFER`**:
    - La opción `SA_RESETHAND` es compatible con la especificación SVr4 del mismo nombre.
    - La opción `SA_NODEFER` es compatible con la de SVr4 a partir del núcleo de Linux versión 1.3.9.

    * Los nombres SA_RESETHAND y SA_NODEFER para compatibilidad con SVr4 están presentes solo en las versiones de la biblioteca 3.0.9 y superiores.
  - **SA_SIGINFO**: Esta opción viene especificada por el estándar POSIX.1b. El soporte para esta opción se añadió en la versión 2.2 de Linux.

📖 Actividad 5.2. Trabajo con las llamadas al sistema `sigsuspend` y `sigprocmask`

📝 **Ejercicio 3.** Escribe un programa que suspenda la ejecución del proceso actual hasta que se reciba la señal SIGUSR1. Consulta en el manual en línea `sigemptyset` para conocer las distintas operaciones que permiten configurar el conjunto de señales de un proceso.

📝 **Ejercicio 4.** Compila y ejecuta el siguiente programa y trata de entender su funcionamiento.

/tarea12.c

```c
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

static int signal_recibida = 0;



static void manejador(int sig)
{
    signal_recibida = 1;
}


int main(int argc, char *argv[])
{
    sigset_t conjunto_mascaras;
    sigset_t conj_mascaras_original;
    struct sigaction act;

    /* Iniciamos a 0 todos los elementos de la estructura act */
    memset (&act, 0, sizeof(act));

    act.sa_handler = manejador;

    if (sigaction(SIGTERM, &act, 0)) {
        perror ("sigaction");
        return 1;
    }

    /* Iniciamos un nuevo conjunto de mascaras */
    sigemptyset (&conjunto_mascaras);
```

```
    /* Añadimos SIGTERM al conjunto de mascaras */
    sigaddset (&conjunto_mascaras, SIGTERM);

    /* Bloqueamos SIGTERM */
    if (sigprocmask(SIG_BLOCK, &conjunto_mascaras, &conj_mascaras_original) < 0) {
       perror ("primer sigprocmask");
       return 1;
    }

    sleep (10);

    /* Restauramos la señal - desbloqueamos SIGTERM */
    if (sigprocmask(SIG_SETMASK, &conj_mascaras_original, NULL) < 0) {
       perror ("segundo sigprocmask");
       return 1;
       }

    sleep (1);

   if (signal_recibida)
       printf ("\nSenal recibida\n");
   return 0;
}
```