sesion5.pdf

Módulo II

2º Sistemas Operativos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

Sesión 5:

Llamadas al sistema para gestión y control de señales

Índice:

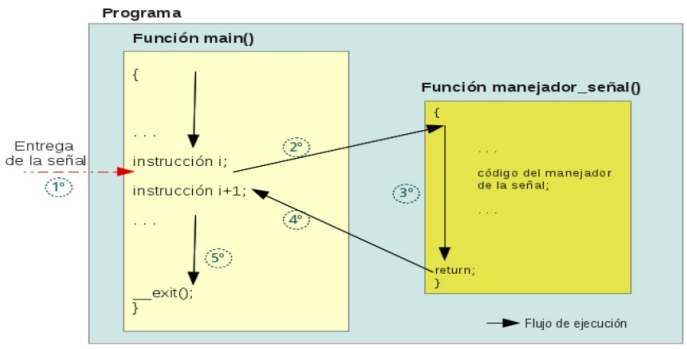
1. Señales

Las **señales** son un **mecanismo básico de sincronización** que indican a los procesos la ocurrencia de algún evento síncrono/asíncrono. Estas señales se envían:

- del núcleo de Linux a los procesos;
- entre procesos.

Las señales se generan cuando **ocurre el evento** y los procesos 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 envía 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 se ve en la imagen siguiente.



Una señal puede estar:

- **depositada**, cuando el proceso inicia una acción en base a ella;
- **pendiente**, si ha sido generada pero todavía no ha sido depositada;
- **ignorada**, es desechada/ignorada por el proceso;
- **bloqueada**, permanece pendiente y será depositada cuando el proceso la desenmascare (la desbloquee).

  Hay que saber diferenciar la ignorada de la bloqueada.

Las señales:

- la **máscara de señales** o **máscara de bloqueo de señales** son las señales bloqueadas de un proceso;
- si una señal **es recibida varias veces** mientras está bloqueada, **se maneja** como si se hubiese recibido **una sola vez**;
- un proceso puede bloquear la recepción de una o varias señales a la vez;
- poseen un **nombre** que comienza por **SIG**, mientras que el resto de los caracteres se relacionan con el **tipo de evento** que representa;
- llevan 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.

La lista de señales y su tratamiento por defecto se puede consultar con **man 7 signal** (o en signal.h). En la tabla siguiente se muestran las señales posibles en POSIX.1:

Símbolo	Acción	Número	Significado
SIGHUP	Term	1	Desconexión del terminal (referencia a la función termio(7) del man). También reanuda los demonios init, httpd e inetd. Esta señal la envía un proceso padre a un proceso hijo cuando el padre finaliza.
SIGINT	Term	2	Interrupción procedente del teclado (<Ctrl+C>)
SIGQUIT	Core	3	Terminación procedente del teclado
SIGILL	Core	4	Excepción producida por la ejecución de una instrucción ilegal
SIGTRAP	Core	5	Punto de interrupción
SIGABRT	Core	6	Señal de aborto procedente de la llamada al sistema abort(3)
SIGIOT	Core	6.	Interrupción de E/S. Es sinónima de SIGABRT
SIGBUS	Core	7	Error en el bus, bad memory access
SIGEMT	Term	-.	Emulación de interrupción // Hace lo mismo que kill -9
SIGFPE	Core	8	Excepción de coma flotante
SIGKILL	Term	9.	Señal para terminar un proceso (no se puede ignorar ni manejar).
SIGUSR1	Term	10	Señal 1 definida por el usuario
SIGSEGV	Core	11	Referencia inválida a memoria
SIGUSR2	Term	12	Señal 2 definida por el usuario
SIGPIPE	Term	13	Tubería rota: escritura sin lectores
SIGALRM	Term	14	Señal de alarma procedente de la llamada al sistema alarm(2)
SIGTERM	Term	15	Señal de terminación
SIGSTKFLT	Term	16	Fallo de pila en el coprocesador (no usada)
SIGCHLD/ SIGCLD	Ign	17	Proceso hijo terminado o parado
SIGCONT	Cont	18	Reanudar el proceso si estaba parado
SIGSTOP	Stop	19.	Parar proceso (no se puede ignorar ni manejar).
SIGTSTP	Stop	20	Parar la escritura en la tty
SIGTTIN	Stop	21	Entrada de la tty para un proceso en background
SIGTTOU	Stop	22	Salida a la tty para un proceso en background
SIGURG	Ign	23	Condición de urgencia en el socket
SIGXCPU	Core	24	El tiempo límite de CPU se ha excedido.
SIGXFSZ	Core	25	El tamaño límite del fichero se ha superado.
SIGVTALRM	Term	26	Alarma de reloj virtual
SIGPROF	Term	27	El temporizador de generación de perfiles expiró
SIGWINCH	Ign	28	Señal para cambiar el tamaño de la ventana
SIGIO/SIGPOLL	Term	29	GIO: la E/S es ahora posible ; POLL: Evento sondeable
SIGPWR	Term	30	Fallo de alimentación
SIGSYS/ SIGUNUSED	Core	31	Mala llamada del sistema

La columna Acción especifica la acción por defecto para la señal, cada uno de ellas significa:

- **term:** terminar el proceso;
- **ign:** ignorar la señal;
- **core:** terminar el proceso y realizar un volcado de memoria;
- **stop:** detener el proceso;
- **cont:** si el proceso está parado, que continúe su ejecución;

2. Llamadas al sistema

2.1 signal

Instala un nuevo manejador de señales a una señal.

```
sighandler_t signal(int s, sighandler_t manejador);
```

- **s**: la señal a la que le vamos a asociar un nuevo manejador;
- **manejador**: la nueva acción a asociar a la señal s. Este manejador puede ser:
  - **SIG_IGN:** con lo que al recibir la señal se ignoraría;
  - **SIG_DFL:** se realiza la acción por defecto de la señal (se puede ver en man 7 signal);
  - **manejador**: nombre de la función que hemos implementado para esa señal, por lo que cuando se recibe la señal, se realiza lo que hay en esta función (manejador).
- devuelve la función previa asignada a la señal s; o SIG_ERR en caso de error.

2.2 kill

Llamada para enviar cualquier señal a un proceso o grupo de procesos.

```
int kill(pid_t pid, int sig);
```

- **pid**: identificador del proceso al que se le va a enviar la señal;
- **sig**: número de la señal a enviar;
- devuelve 0 si ha habido éxito; -1 en caso de error y modifican errno con el tipo de error.

A tener en cuenta:

- Si **pid > 0**; se envía la señal sig al proceso pid.
- Si **pid = 0**; sig se envía a cada proceso en el grupo de procesos del proceso actual.
- Si **pid = -1**; se envía sig a cada proceso excepto al primero, desde los números más altos en la tabla de procesos hasta los más bajos.
- Si **pid < -1**; se envía sig a cada proceso en el grupo de procesos -pid.
- Si **sig = 0**, no se envía ninguna señal y se realiza una comprobación de errores.

Haciendo kill -l en la terminal, nos muestra el conjunto de señales que hay.

2.3 sigaction

Llamada para cambiar la acción tomada por un proceso cuando recibe una señal.

```
int sigaction(int sig, const struct sigaction *act, struct sigaction *old_act);
```

- **sig**: número de la señal a enviar, excepto SIGKILL y SIGSTOP;
- **act**: si no es NULL, es la nueva acción que se instala para la señal sig;
- **old_act:** si no es NULL, la acción de la señal sig antes de instalar act, se guarda en esta estructura;
- devuelve 0 si ha habido éxito; -1 en caso de error y modifican errno con el tipo de error.

Le asigna manejador de señales (función implementada por nosotros) a una señal cambiándole su acción por defecto, por lo que cuando se envía dicha señal se realiza lo que hay implementado en la función asignada.

Estructura sigaction

```
struct sigaction {
        void (*sa_handler)(int);
        void (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t sa_mask;
        int sa_flags;
        void (*sa_restorer)(void);          //está obsoleto y no debería utilizarse
}
```

donde cada parámetro es:

- **sa_handler**: especifica la acción de la señal sig. Puede tomar las siguientes acciones:
  - **SIG_DFL** para la acción predeterminada;
  - **SIG_IGN** para ignorar la señal;
  - o un puntero a una **función manejadora** para la señal, una función implementada por nosotros.
- **sa_mask:** establece una **máscara de señales** que deberían bloquearse durante la ejecución del manejador de la señal. A menos que SA_NODEFER o SA_NOMASK estén desactivadas, la señal que lance el manejador será bloqueada. Las siguientes funciones asignan valor a esta variable:
  - **int sigemptyset(sigset_t *set)**: inicializa a vacío el conjunto de señales set, es decir, la máscara de señales set la deja vacía (0 tiene éxito, -1 error);
  - **int sigfillset(sigset_t *set):** inicializa el conjunto set con todas las señales, es decir, la máscara de señales set la llena con todas las señales (0 si tiene éxito, -1 error);
  - **int sigismember(const sigset_t *set, int sig):** determina si una señal sig pertenece a un conjunto de señales set, si una señal está en la máscara set (0 si no está en el conjunto, 1 si está dentro);
  - **int sigaddset(sigset_t *set, int sig):** añade la señal sig a un conjunto de señales set previamente inicializado (0 si tiene éxito, -1 error);
  - **int sigdelset(sigset_t *set, int sig):** elimina una señal signo de un conjunto de señales set (0 si tiene éxito, -1 error);
- **sa_flags**: especifica un conjunto de opciones para modificar el comportamiento del proceso de manejo de señales. Se forma por el OR a cero o más de las siguientes constantes:
  - **SA_NOCLDSTOP**: si sig es SIGCHLD, indica al núcleo que el proceso no desea recibir notificación cuando los procesos hijos se paren (cuando los procesos hijos reciban 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**: hace que ciertas llamadas reinicien la ejecución cuando son interrumpidas por una señal. Proporciona un comportamiento compatible con la semántica de señales de BSD;
  - **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, por lo que hay que configurar sa_sigaction en vez de sa_handler.

Estructura siginfo_t

```
struct 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 */
        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 */
}
```

Estructura sigset_t

Es un tipo que representa la estructura de un conjunto de señales. Según POSIX, debe ser un entero o una estructura.

2.4 Ejercicio 1

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 a un proceso cuyo identificador de proceso es PID:
        0: SIGTERM      1: SIGUSR1      2: SIGUSR2
SINTAXIS: envioSignal [012] <PID> */
#include <sys/types.h>          #include <limits.h>
#include <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>

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[]){
        struct sigaction sig_USR_nact;
```

```
        if(setvbuf(stdout, NULL, _IONBF,0)){
                perror("\nError en setvbuf");
        }

        //Inicializar la estructura sig_USR_na para especificar la nueva acción para la señal.
        sig_USR_nact.sa_handler= sig_USR_hdlr;

        //'sigemptyset' inicia el conjunto de señales dado al conjunto vacío.
        sigemptyset (&sig_USR_nact.sa_mask);
        sig_USR_nact.sa_flags = 0;

        //Establecer mi manejador particular de señal para SIGUSR1
        if( sigaction(SIGUSR1,&sig_USR_nact,NULL) <0){
                perror("\nError al intentar establecer el manejador de señal para SIGUSR1");
                exit(-1);
        }

        //Establecer mi manejador particular de señal para SIGUSR2
        if( sigaction(SIGUSR2,&sig_USR_nact,NULL) <0){
                perror("\nError al intentar establecer el manejador de señal para SIGUSR2");
                exit(-1);
        }
        for(;;){ }
}
```

El programa envioSignal.c:

- recibe un identificador de señal y el pid de un proceso;
- envía la señal recibida por argumento al proceso pid;
- si la señal es 0, termina el proceso pid;
- si la señal es 1 ó 2, se va al manejador de señal implementada en reciboSignal y muestra un mensaje por pantalla;
- para cualquier otra señal, imprime por pantalla un mensaje de que no se puede enviar la señal.

El programa reciboSignal.c:

- implementa un manejador de señal, en el que si recibe la señal SIGUSR1 ó SIGUSR2, imprime por pantalla que ha recibido la señal.

Compilamos y ejecutamos:

```
$ ./reciboSignal &                          //se queda esperando a la señal
        [1] 34058
$ ./envioSignal 1 34058                     //le enviamos la señal 1
        Recibida la señal SIGUSR1
$ ./ envioSignal 2 34058                    //le enviamos la señal 2
        Recibida la señal SIGUSR2
$ ./ envioSignal 3 34058                    //le enviamos la señal 3
        No puedo enviar ese tipo de señal
$ ./ envioSignal 0 34058                    //le enviamos la señal 0 para que termine
        [1]+  Terminado          ./reciboSignal
```

2.5 Ejercicio 2

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.

```
#include <sys/types.h>
#include <limits.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
#define SIG 31

//vector de señales
static int contador_sig[SIG];

/*función para contar las señales que se recibe
   - s es el número de la señal*/
void contador(int s){
        //incrementamos el número de veces que se ha recibido dicha señal
        contador_sig[s-1]++;

        //imprimimos por pantalla el número de veces que se ha recibido la señal
        printf("La señal %i se ha recibido %i veces\n", s, contador_sig[s-1]);
}

int main(int argc, char * argv[]){
        //estructura para trabajar con la llamada sigaction
        struct sigaction cont;

        //para que envíe los mensajes conforme los reciba y no en bloques
        if(setvbuf(stdout, NULL, _IONBF, 0)){
                perror("\nError en setvbuf");
        }

        /*Inicializamos los campos de cont (sa_handler, sa_mask y sa_flags*/
        //especificamos que la función manejadora de señales es contador
        cont.sa_handler = contador;

        //inicializamos a vacío el conjunto de señales
        sigemptyset(&cont.sa_mask);

        //no modificamos el comportamiento del proceso
        cont.sa_flags = 0;

        /*inicilizamos el vector de señales a 0*/
        for(int i = 0; i < SIG; i++){
                contador_sig[i] = 0;
        }

        /*establecemos el manejador contador para cada señal que se reciba con sigaction*/
        for(int i = 1; i <= SIG; i++){
```

```
                //si la señal es SIGKILL o SIGSTOP, no le asignamos manejador
                if(i != SIGKILL && i != SIGSTOP){
                        //asignamos el manejador cont a la señal i
                        //si hay error, se imprime por pantalla
                        if( sigaction(i, &cont, NULL) < 0){
                                perror("ERROR al intentar establecer manejador de señal\n");
                                exit(-2);
                        }
                }
        }

        //esperamos indefinidamente a recibir las señales
        for(;;){}

        return 0;
}
```

Ejecutamos el programa en background

```
$ ./contador &
        [1] 77145
```

Le vamos enviando señales y vemos cómo las contabiliza

```
$ kill -2 77145
        La señal 2 se ha recibido 1 veces
$ kill -3 77145
        La señal 3 se ha recibido 1 veces
$ kill -4 77145
        La señal 4 se ha recibido 1 veces
$ kill -5 77145
        La señal 5 se ha recibido 1 veces
$ kill -5 77145
        La señal 5 se ha recibido 2 veces
$ kill -5 77145
        La señal 5 se ha recibido 3 veces
$ kill -18 77145
        La señal 18 se ha recibido 1 veces
$ kill -18 77145
        La señal 18 se ha recibido 2 veces
$ kill -18 77145
        La señal 18 se ha recibido 3 veces
$ kill -9 77145                 //con esta señal finalizamos el proceso
        [1]+  Terminado (killed)      ./contador
```

2.6 sigprocmask

Llamada para cambiar la **lista de señales bloqueadas**, es decir, **la máscara de señales**. No es posible bloquear SIGKILL ó SIGSTOP con esta llamada; el núcleo no tendrá en cuenta los intentos.

```
int sigprocmask(int tipo, const sigset_t *set, sigset_t *old_set);
```

- **tipo**: es el tipo de cambio que se hace sobre la máscara set; puede tomar los siguientes valores:
  - **SIG_BLOCK:** añadimos a la máscara de señales del proceso que llama, la máscara de señales pasada en set; es decir, al conjunto actual de señales bloqueadas del proceso le añadimos el conjunto de señales que hay en set;
  - **SIG_UNBLOCK:** a la máscara de señales del proceso actual le quitamos las señales que estén en la máscara de señales set; es decir, al conjunto de señales bloqueadas del proceso le quitamos las señales que estén en set. Es posible intentar el desbloqueo de una señal que no está bloqueada;
  - **SIG_SETMASK:** la máscara del proceso se establece a set; es decir, el conjunto de señales bloqueadas del proceso se inicializa al conjunto de señales de set.
- **set**: conjunto de señales que cambia su función según el parámetro anterior; si es NULL, esta llamada se usa como consulta;
- **old_set:** conjunto de señales antes de ser modificadas según el tipo; si es NULL, no se devuelve el conjunto anterior;
- devuelve 0 en caso de éxito, -1 en caso de error.

2.7 sigpending

Llamada al sistema para **atender las señales pendientes** de un proceso.

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

- **set:** la llamada devuelve en este argumento las señales pendientes del proceso que la invoca;
- devuelve 0 si ha habido éxito; -1 en caso de error y modifican errno con el tipo de error.

2.8 sigsuspend

Llamada para que un proceso espere por una señal.

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

- **mask:** máscara de señales por las que un proceso NO espera; es decir, el proceso se suspende hasta que llegue una señal que NO se encuentra en mask;
- devuelve -1 en caso de error y modifica errno con el tipo de error.

Esta llamada reemplaza **temporalmente** la **máscara de señal del proceso actual** por **mask** y luego suspende el proceso hasta la entrega de una señal que no esté en mask.

- Si la señal recibida termina el proceso, sigsuspend() no regresa.
- Si se detecta la señal, sigsuspend() regresa después de que regrese el manejador de señales, y la máscara de señal se restaura al estado anterior a la llamada a sigsuspend().

No es posible bloquear SIGKILL o SIGSTOP; especificar estas señales en la máscara, no tiene ningún efecto en la máscara de señal del proceso.

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.

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

void manejador(int i){
        printf("Acaba de desbloquearse el proceso.\n");
}

int main(){
        sigset_t mascara;
        struct sigaction tarea;

        //inicializamos la máscara
        sigemptyset(&tarea.sa_mask);

        //le asignamos manejador
        tarea.sa_handler = manejador;

        //le asignamos manejador a SIGUSR1
        sigaction(SIGUSR1, &tarea, NULL);
        tarea.sa_flags = 0;

        //vacíamos y añadimos todas las señales a máscara
        sigemptyset(&mascara);
        sigfillset(&mascara);

        //eliminamos SIGUSR1 de la máscara
        sigdelset(&mascara, SIGUSR1);

        //suspende el proceso hasta que se envía alguna de las señales que NO están en la máscara
        sigsuspend(&mascara);

        return 0;
}
```

Compilamos y ejecutamos:

```
./tarea &
        [1] 79694
$ kill -17 79694            //cualquier señal que enviemos la ignora
$ kill -18 79694
$ kill -1 79694
$ kill -3 79694
$ kill -SIGUSR1 79694       //cuando enviamos la señal SIGUSR1, reanuda el proceso y lo finaliza
        Acaba de desbloquearse el proceso.
        [1]+  Colgar (hangup)       ./tarea
```

La última línea de Colgar (hangup) se debe a que cuando ha reanudado el proceso después de SIGUSR1, ha recibido todas las señales bloqueadas, y una de ellas ha sido la señal -1, de SIGHUP.

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

```
//tarea12.c
#include <signal.h>      #include <stdio.h>
#include <string.h>      #include <unistd.h>

static int signal_recibida = 0;

static void manejador{
        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 ("\nSeñal recibida\n");
        return 0;
}
```

El programa:

- bloquea la señal SIGTERM durante 10s;
- se desbloquea dicha señal;
- si durante los 10s de espera ha recibido dicha señal SIGTERM, muestra por pantalla que la ha recibido.
- Cuando se recibe SIGTERM, se ejecuta el manejador de señales que se le ha asociado, que establece signal_recibida a 1.

Ejecución:

- Cuando enviamos la señal SIGTERM:

```
$ ./tarea &
        [1] 81151
$ kill -SIGTERM 81151
$                                   //esperamos un poco
        Señal recibida
        [1]+  Hecho                 ./tarea
```

- Cuando no enviamos la señal SIGTERM:

```
$ ./tarea &
        [1] 81236                   //esperamos un poco
$
        [1]+  Hecho                 ./tarea
```

4. Preguntas de repaso

1. **¿Cómo podemos enviar a un proceso expresamente la señal que hemos bloqueado en la tarea11 por terminal?**

   $ kill -señal PID

   PID del proceso que hemos ejecutado de la tarea11.

2. **¿Cómo podemos saber el PID del proceso ejecutado?**

   $ ps -e | grep nombreEjecutable → ps -e | grep tarea11

   $ ps -aux | grep nombreEjecutable → ps -aux | grep tarea11

   Podemos ejecutarlo en segundo plano con & (./tarea11 &), con lo que nos muestra el PID cuando se ejecuta.

3. **¿Cómo podríamos implementar que al recibir una señal, simplemente muestre por pantalla que ha recibido dicha señal?**

   Con el manejador de señales. Creamos una función que será el manejador, y en el main se le asigna a la señal dicha función.

4. **¿Cómo podríamos implementar que un proceso le mande una señal a un proceso?**

   Con la orden kill(señal, PID) en el programa que envía la señal y el PID del otro proceso.

   El ejercicio 1 de la sesión 5 lo muestra.

5. **¿Cómo podríamos hacer lo mismo que lo anterior pero sin pasarle por argumento el PID del proceso al que se le va a mandar la señal?**

   En un mismo programa, realizando un fork() para crear un proceso hijo.

6. **El programa anterior creando un hijo, no funciona, ¿por qué puede ser?**

   Porque el padre no espera al hijo, envía las señales antes de que se ejecute el hijo, por lo que habría que realizar un wait o un sleep para que le dé tiempo al hijo a ejecutarse y recibir las señales.