Escuela Politécnica Superior de Elche

SISTEMAS INFORMÁTICOS EN TIEMPO REAL
2º Ingeniería Industrial

SEÑALES POSIX

Luis Miguel Jiménez
Rafael Puerto

Departamento de Ingeniería de Sistemas Industriales
Área de Ingeniería de Sistemas y Automática

13 SEÑALES

Se resumen a continuación los aspectos fundamentales asociados al manejo de señales, centrándose en los mecanismos básicos ofrecidos por la implementación POSIX *SUN-Solaris*. Estudiaremos las funciones permiten:

- Configurar una señal
- Ejecutar tareas ante eventos asíncronos

13.1 ¿Qué es una Señal?

Una **señal** es una notificación por software a un proceso/thread de la ocurrencia de un evento.

Existe cierto paralelismo con el uso de **interrupciones**, aunque se trata de mecanismos totalmente diferentes. En el caso de una interrupción, se implementa mediante el propio hardware del computador de forma que, ante un evento interno o externo (señalado mediante la activación de alguna entrada de la CPU) se interrumpe la instrucción actualmente en curso, para comenzar a ejecutar una nueva.

En el caso de una **señal**, la aparición del evento es controlada por el software del S.O. decidiendo cuando se interrumpe el proceso o thread que tiene actualmente los recursos, para activar la tarea asociada a la señal. El soporte software permite que una señal pueda ser encolada o enmascarada. De igual forma permite asociar señales independientes a cada proceso o thread en ejecución. Se trata por tanto de un mecanismo más flexible y potente.

Es conveniente distinguir las diferentes estados en los que se puede encontrar una señal:

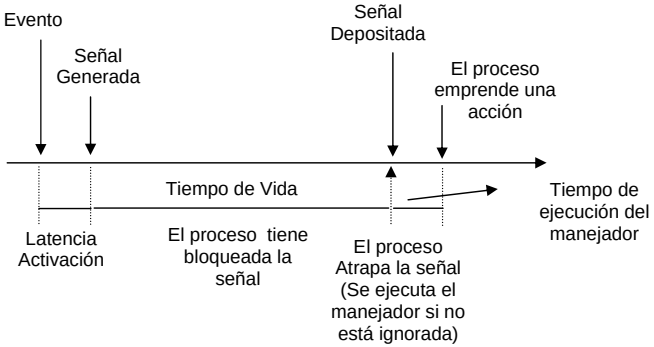
1. Una señal está asociada a un evento, por lo que cuando dicho evento se produce se dice que la señal se ha *generado*.
2. Se dice que la señal está *depositada* cuando el proceso asociado emprende una acción en base a ella.
3. El *tiempo de vida* de una señal es el intervalo entre la generación y el depósito de ésta.

4. Se dice que una señal está *pendiente* si ha sido generada pero todavía no está depositada.

5. Un proceso *atrapa* una señal si éste ejecuta el manejador de señal cuando se deposita.

6. Una señal puede ser *ignorada* por el proceso, es decir que no sea ejecutado ningún manejador al ser depositada. Destacar que aunque no se ejecute ningún manejador la señal si que llega al proceso y éste puede determinar alguna acción en función de ello. El que una señal sea atrapada o ignorada es especificado en la configuración de la señal.

7. La acción que se emprende cuando se genera una señal depende de la *máscara* de la señal. La máscara contiene una lista de señales que en un determinado momento están *bloqueadas*. Si se genera una señal y está bloqueada no se pierde, queda pendiente de ser depositada hasta que sea desbloqueda.



A lo largo de este capítulo se presentarán dos tipos de señales:

- **Señales estándar**: son las señales definidas en el estándar POSIX 1. Utilizan las funciones originales desarrolladas en UNIX.

- **Señales de Tiempo Real**: Definidas en POSIX 1.b extienden la funcionalidad permitiendo el manejo de prioridades, el paso de información a los manejadores y el encolado de las mismas (permite que varios eventos de la misma señal estén en espera y puedan ser despositados).

13.2 ¿Cómo se identifica una señal?

El identificador de una señal es un número entero, pero para facilitar su utilización todas las señales tienen asociado un nombre simbólico que comienza con el prefijo `SIG`. Los nombres de las señales están definidos en el archivo `<sys/signal.h>`. Realmente basta con incluir en nuestro programa la cabecera `<signal.h>` ya que ésta incluye a la anterior. A continuación se muestra un listado de las señales predefinidas:

SÍMBOLO	SIGNIFICADO
SIGHUP	Hangup
SIGINT	Interrupt (rubout)
SIGQUIT	Quit (ASCII FS)
SIGILL	illegal instruction (not reset when caught)
SIGTRAP	trace trap (not reset when caught)
SIGIOT	IOT instruction
SIGABRT	used by abort, replace SIGIOT in the future
SIGEMT	EMT instruction
SIGFPE	floating point exception
SIGKILL	kill (cannot be caught or ignored)
SIGBUS	bus error
SIGSEGV	segmentation violation
SIGSYS	bad argument to system call
SIGPIPE	Write on a pipe with no one to read it
SIGALRM	Alarm clock
SIGTERM	software termination signal from kill
SIGUSR1	**User defined signal 1**
SIGUSR2	**User defined signal 2**
SIGCLD / SIGCHLD	Child status change Child status change alias (POSIX)
SIGPWR	power-fail restart
SIGWINCH	window size change
SIGURG	urgent socket condition
SIGPOLL / SIGIO	pollable event occurred socket I/O possible (SIGPOLL alias)
SIGSTOP	stop (cannot be caught or ignored)
SIGTSTP	user stop requested from tty
SIGCONT	stopped process has been continued
SIGTTIN	background tty read attempted
SIGTTOU	background tty write attempted
SIGVTALRM	virtual timer expired
SIGPROF	profiling timer expired
SIGXCPU	exceeded cpu limit
SIGXFSZ	exceeded file size limit
SIGWAITING	process's lwps are blocked
SIGLWP	special signal used by thread library
SIGFREEZE	special signal used by CP
SIGTHAW	special signal used by CPR
SIGCANCEL	thread cancellation signal used by libthread
SIGLOST	resource lost (eg, record-lock lost)
SIGRTMIN	**first (highest-priority) realtime signal**
SIGRTMAX	**last (lowest-priority) realtime signal**

Tabla 13.1 Listado de Señales (Sun-Solaris)

Como se puede observar la mayoría de las señales predefinidas están asociados a eventos del propio Sistema Operativo. Su uso está fijado y no puede cambiarse (por ejemplo si generamos una señal **SIGKILL** a un proceso este terminará ya que todo proceso incluye por defecto el manejo de las señales estándar. En la tabla se han destacado en negrita dos grupos de señales:

- Las señales definibles por el usuario **SIUSR1**, **SIGUSR2** pueden ser utilizadas por los procesos de nuestra aplicación libremente para intercambiar eventos, o asociarlas a temporizadores.

- Las señales **SIGRTMIN** y **SIGRTMAX** representan un rango de valores para señales asociadas a la gestión de eventos en sistemas de tiempo real, atendidos mediante un mecanismo de prioridades. Se pueden utilizar libremente en las aplicaciones del usuario.

13.3 ¿Cómo se genera una señal?

Las señales son un mecanismo software, por lo que es el propio sistema operativo el que se encarga de su generación en función del evento asociado. Estos eventos pueden tener un origen externo, como la activación de un pin de la CPU o de un determinado registro o puerto del computador. También pueden estar asociados a interrupciones hardware del propio computador, como por ejemplo cuando el contador de un reloj alcanza cierto valor.

Aparte de este tipo de eventos, una señal puede ser generada por los propios procesos que se están ejecutando en el computador (software). Permite de este modo un mecanismo para que los diferentes procesos se comuniquen entre sí la aparición de eventos que determinan la ejecución del programa.

Vamos a estudiar dos métodos de generación de señales:

13.3.1 Generación una señal desde otro proceso:

Para generar una señal estándar en un proceso determinado desde otro proceso podemos utilizar la función ***kill()***(ver prototipo en la parte inferior). El primer parámetro ***pid*** identifica al proceso que recibirá la señal. En segundo parámetro ***sig*** identifica la señal a mandar. La función ***sigsend()*** actúa de forma similar pero permite una gestión más potente de los procesos destinatarios de la señal.

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
int sigsend(idtype_t idtype, id_t id, int sig);
```


Otra forma sencilla de enviar una señal a un proceso es utilizar el comando *kill [-signal] pid* desde la consola. Aunque el nombre *kill* parezca extraño para enviar una señal, hay que tener en cuenta que muchas señales tiene como acción detener o terminar un proceso.

Las **señales de tiempo real**, aparte del hecho de ser generadas y depositadas, permiten adicionalmente enviar un valor (entero) al proceso destinatario y al manejador de la señal. Las funciones *kill()* y *sigsend()* no permiten especificar este valor por lo que si se desea enviar tal valor debe usarse la función ***sigqueue()*** permite generar una señal especificando un valor entero. Esta última función incorpora adicionalmente la posibilidad de que la señal sea encolada (que se quede en espera con una prioridad) si no puede ser atendida por el proceso (POSIX1.b).

```
int sigqueue(pid_t pid, int signo, const union sigval value);

union sigval {
        int             sival_int;  /* integer value */
        void            *sival_ptr; /* pointer value */
     };
```

13.3.2 Generación una señal desde un temporizador:

En un aplicación de tiempo real, el tipo de generación que más nos interesa es la activación asociada a relojes contadores (temporizadores). El concepto de temporizador se desarrolla en el siguiente apartado, pero quedémonos con la idea de que se trata de un registro que se decrementa de forma regular según un reloj. Al alcanzar el valor cero expira y genera una señal.

Dentro de la posible configuración del temporizador se puede especificar que al expirar vuelva a al valor original y comience el ciclo de nuevo, permitiendo de este podo generar señales de forma periódica.

En el capítulo 9 se estudian las funciones asociadas para la generación de señales mediante temporizadores.

13.4 ¿Quién recibe una Señal?

Las señales van dirigidas a procesos, pero dentro de ellos pueden existir varios threads. Las señales, por tanto, pueden ir dirigidas a un thread concreto o a todos, esto depende del tipo de señal. Básicamente existen dos tipos de señales:

1. **Señales síncronas**: son aquellas que son generadas por la ejecución del código, y por tanto son generadas por un thread concreto. Por ejemplo SIGBUS o SIGFPE son generadas por errores del código ejecutado. En este caso la señal puede ser depositada únicamente por el thread que generó el evento

2. **Señales asíncronas**: el resto de señales producidas por llamas explícitas (kill) o por eventos no asociados al código en ejecución. En este caso la señal va dirigida a todos los threads del proceso. Pero una señal solo puede ser depositada por un thread así que surge la pregunta de *¿cuál será el thread que deposite la señal?*. Pues bien, la recibirá uno cualquiera que cumpla una de las siguientes condiciones:

   - El thread está bloquedado esperando la señal. Si existen varios habrá una cola de prioridades.

   - El thread no incluye en su **máscara de bloqueo** la señal en cuestión (la señal no está bloqueada por el thread)

Generalmente en una aplicación se escogen uno o varios threads para atender aquellas señales importantes quedando dichos threads *suspendidos* a la espera de que se genere el evento de forma asíncrona. De este modo aseguramos cual va a ser el thread, que de forma unívoca, va a atender cada señal.

13.5 Configuración de una señal

En primer lugar debemos recordar que las señales asíncronas (que son las que utilizaremos) van dirigidas a todos los threads del proceso y depende de la configuración de cada uno de ellos, en cual será depositada. Por ello debemos tener en cuenta que van a existir dos aspectos diferentes de configuración y gestión de señales:

- Aquellos aspectos que afectan únicamente al thread que las llama: irán ubicadas generalmente en cada uno de los manejadores del thread (función indicada en la creación de cada thread).

- Aquellos aspectos comunes a todos los threads: se ubican generalmente en el thread principal.

La configuración de una señal tiene dos partes fundamentales:

1. Decidir entre **atrapar** la señal, es decir asignar un manejador para la señal (función que se ejecuta cuando es depositada), o bien **ignorar** la señal. Se trata de un aspecto común para todos los threads

2. Establecer el bloqueo de la señal (la señal queda pendiente de ser depositada). Es un aspecto de configuración independiente para cada thread

13.5.1 Bloqueo de una señal

El bloqueo de una señal supone que dicha señal quede pendiente de depósito en el caso de que sea generada, es decir la señal no se pierde sino que su atención es postergada. El bloqueo de una señal por parte de un proceso es específico de cada thread del proceso. Por lo tanto, dentro de un proceso pueden existir threads que hayan bloqueado la señal y threads que no la hayan bloqueado. Estos últimos son los únicos que pueden depositar la señal (recordemos que una señal solo será depositada por uno de los threads del proceso).

La gestión del bloqueo de señales se realiza mediante lo que se denomina **máscara de señal** esta máscara se maneja mediante una estructura de tipo ***sigset_t*** almacenada por el S.O. para cada thread. La máscara de señal indica la configuración de cada señal (bloqueada o no bloqueada). Adicionalmente podemos definir nuevos **conjuntos de señales** de tipo ***sigset_t*** que nos permitirán activar o desactivar grupos de señales en cada momento.

El manejo de estos **conjuntos** se debe realizar con las funciones suministradas y básicamente lo que realizan es especificar las señales sobre las que queremos ejecutar la acción correspondiente. Esta acción depende de la función en la que se use dicho conjunto (por ejemplo ***sigprocmask()***). Las funciones de manejo básico son las siguientes:

sigemptyset(): inicializa un conjunto excluyendo todas las señales
sigfillset(): inicializa un conjunto incluyendo todas las señales
sigaddset(): incluye la señal indicada en el conjunto
sigdelset(): excluye la señal indicada en el conjunto

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
```

Una vez especificado el conjunto de señales a través de un conjunto ***sigset_t***, podemos ejecutar su bloqueo o desbloqueo.

Disponemos de dos funciones para el bloqueo o desbloqueo de una señal: ***sigprocmask()*** realiza el bloqueo de la señal especificada para un proceso(en algunas implementaciones afecta a todos los threads, mientras que en otras solo afecta al thread que lo ejecutó [SUN-Solaris]). La función ***pthread_sigmask()*** bloquea únicamente al thread que la llama. Destacar que si un thread crea a otro, éste último

hereda el bloqueo de las señales. En todo caso para programas basados en múltiples threads se recomienda el uso de la segunda función.

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

int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset);
```

Los parámetros de las dos funciones son los mismos:

- ***how***: indica la acción a realizar: **SIG_BLOCK**, (para bloquear un conjunto de señales), **SIG_UNBLOCK** para desbloquearlas, **SIG_SETMASK** copia el conjunto dato en ***set*** en la máscara.
- ***set***: es un puntero al conjunto de señales que queremos bloquear o desbloquear (tal como se describió anteriormente)
- ***oset***: si no es NULL almacena el valor de la máscara anterior a la operación

A continuación se muestra un ejemplo de bloqueo y desbloqueo de una señal:

```
#include <signal.h>

main()
{
      sigset_t sigset;  // conjunto de señales

      // bloquea la señal
      sigemptyset(&sigset);   // crea una máscara vacía
      sigaddset(&sigset, SIGUSR1);  // añade la señal
      sigprocmask (SIG_BLOCK, &sigset, NULL); // bloquea la señal

      // desboquea la señal
      sigemptyset(&sigset);
      sigaddset(&sigset, SIGUSR1);
      sigprocmask(SIG_UNBLOCK, &sigset, NULL); //desbloquea la señal
}
```

Nota: las aplicaciones que utilizan señales POSIX requieren ser compiladas con la librería posix4 (`-lposix4`)

Asignación de un manejador de señal

Para asignar un manejador para una señal concreta se utiliza la función ***sigaction()***. Un detalle importante es que, salvo ciertas señales (síncronas), generalmente las señales van dirigidas al proceso aunque este esté formado por varios threads. De esta forma, la configuración de la señal es común para todos los threads y solo se realiza una vez.

La función ***sigaction()*** nos permite tres opciones:

1. Ignorar la señal: no se ejecutaría ningún manejador (a pesar de ello la señal si que llegaría al proceso que podría ejecutar una acción)
2. Asignar un manejador por defecto
3. Asignar nuestro propio manejador

Para evitar que se deposite una señal mientras se está configurando se debe, en primer lugar, bloquear la señal en todo el proceso antes de llamar a la función ***sigaction()***. Una vez configurada se procederá a desbloquear dicha señal. Una buena norma es configurar la señal antes de crear los threads, de forma que el bloqueo sea global.

```
int sigaction(int sig, const struct sigaction *act,
                       struct sigaction *oact);
struct sigaction {
      int sa_flags;
      union {
            void (*_handler)(int);
            void (*_sigaction)(int, siginfo_t *, void *);
      }     _funcptr;
      sigset_t sa_mask;
      int sa_resv[2];
};
#define     sa_handler  _funcptr._handler
#define     sa_sigaction      _funcptr._sigaction
```

El primer parámetro ***sig*** especifica la señal que vamos a configurar (ver apartado 3.2). El segundo parámetro ***act*** es un puntero a una estructura donde debemos especificar la configuración deseada. Existen dos tipos configuraciones diferentes para esta estructura. El uso de una u otra viene especificado por el campo ***sa_flags*** :

Señales Estándar

```
struct sigaction
{
   int   sa_flags;
   void (*sa_handler)(int);
   sigset_t   sa_mask;
}
```

Señales de Tiempo Real

```
struct sigaction
{
   int   sa_flags;
   void (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t   sa_mask;
}
```

- **Estructura estándar**: es la más antigua y está soportada por todas las implementaciones POSIX1. El thread que deposita la señal y el manejador de dicha señal, reciben únicamente el número de la señal que se ha generado. Tiene tres campos importantes:
  - ***sa_handler***: se trata de un puntero a una función (manejador de la señal). La función tienen un único parámetro entero que indica el número de la señal. Existen dos valores especiales para el parámetro *sa_handler*:
    - **SIG_DFL**: asigna un manejador por defecto.
    - **SIG_IGN**: ignora la señal (no se ejecuta ningún manejador)
  - ***sa_mask*** : especifica la máscara con las señales adicionales que deben ser bloqueadas durante la ejecución del manejador (cuando la señal es depositada). Normalmente asignaremos una máscara vacía.
  - ***sa_flags*** : especifica opciones especiales. Normalmente será NULL

- **Estructura extendida**: esta especificada en POSIX1.b y permite proporcionar, tanto al thread que deposita la señal como al manejador de la señal, información adicional (precisa activar el flag SA_SIGINFO).Tiene tres campos importantes:
  - ***sa_sigaction***: se trata de un puntero a una función (manejador de la señal). La función tienen tres parámetros:
    - ***int*** : identificador de la señal.
    - ***sig_info_t ****: puntero a una estructura que incluye información adicional sobre la señal. Dispone, al menos, de los siguientes campos
      - ***int si_signo***: número de señal
      - ***int si_code***: causa de la señal
      - ***union sigval si_value***: valor que se le pasa al manejador de la señal (por ejemplo desde un temporizador, o utilizando la función sigqueue())
    - ***void ****: puntero genérico (Actualmente no está definido su uso).

    Como en el caso anterior existen dos valores especiales:
    - **SIG_DFL**: asigna un manejador por defecto.
    - **SIG_IGN**: ignora la señal (no se ejecuta ningún manejador)

  - ***sa_mask*** : especifica la máscara con las señales adicionales que deben ser bloqueadas durante la ejecución del manejador (cuando la señal es depositada). Normalmente asignaremos una máscara vacía.
  - ***sa_flags*** : debe activar al menos el flag SA_SIGINFO indicando el uso de información adicional

El parámetro ***oact*** devuelve la configuración previa a la ejecución de ***sigaction()***. Este parámetro generalmente se pone a NULL, indicando que no devuelva dicha información.

Veamos un ejemplo de configuración de una señal:


```
#define _REENTRANT
#include <pthread.h>
#include <signal.h>

void ManejadorSig ( int signo, siginfo_t *info, void *context);

main()
{
      struct sigaction act;   // manejador señal
      sigset_t sigset;  // conjunto de señales

      // Primero bloquea la señal
      sigemptyset(&sigset);   // crea una máscara vacía
      sigaddset(&sigset, SIGRTMAX); // añade la señal
      pthread_sigmask (SIG_BLOCK, &sigset, NULL); // bloqueo

      // Configura la señal
      act.sa_sigaction = ManejadorSig;    // manejador de la señal
      sigemptyset(&(act.sa_mask));        // máscara vacía
      act.sa_flags = SA_SIGINFO;          // Modo extendido

      if(sigaction(SIGRTMAX, &act, NULL)<0)     // configuración
            perror("Sigaction fallado");

      // desboquea la señal
      sigemptyset(&sigset);
      sigaddset(&sigset, SIGRTMAX);
      pthread_sigmask (SIG_UNBLOCK, &sigset, NULL); //desbloqueo

      while(1)
      {
            // espera indefinidamente la aparición de la señal
            // desde la consola se puede enviar una señal al proceso
            //     kill -s <SIGNUM> <PID>
      }

}

/* Manejador de la Señal */
void ManejadorSig( int signo, siginfo_t *info, void *context)
{
      printf("Soy el manejador de la señal # %d Valor: %d ",
            info->si_signo, info->si_value.sival_int);

      printf("Code: (%d) ",info->si_code);
      if( info->si_code == SI_USER)
            printf("SI_USER \n" );
      else if( info->si_code == SI_TIMER )
            printf("SI_TIMER \n" );
      else if( info->si_code == SI_QUEUE )
            printf("SI_QUEUE \n" );
      else if( info->si_code == SI_ASYNCIO )
            printf("SI_ASYNCIO \n" );
      else if( info->si_code == SI_MESGQ )
            printf("SI_MESGQ \n" );
}
```

13.6 Espera de señales

Una forma de ejecutar una tarea ante la ocurrencia de un evento asíncrono, es configurar una señal con un manejador que incluya el código de la tarea a ejecutar. Cuando la señal sea atrapada en un thread (basta que exista uno que no tenga bloqueda dicha señal) se ejecutaría el manejador de la señal.

El problema radica en que el código de un manejador de señal debe ser **reentrante**, es decir, que se comporte correctamente cuando se activa de nuevo la señal antes de terminar la ejecución de la función. Muchas funciones y llamadas al sistemas son **no reentrantes** por lo que no pueden usarse en un manejador de señal.

Como norma general, los manejadores de señal deben ser rutinas muy cortas por lo que generalmente no es el sitio adecuado para ubicar la tarea que queremos ejecutar de forma asíncrona.

La estrategia correcta es situar el código de la tarea en un **thread**. Dicho thread debe estar *dormido* (**suspendido**) esperando la activación de la señal, poniéndose a ejecutar dicho código cuando la señal sea depositada. Este thread será el único que tendrá desbloqueada la señal con lo que garantizamos que, cuando se genere la señal, será capturada siempre por él. El manejador de la señal (si no es ignorado) seguiría ejecutándose también, por lo que pondremos uno vacío o el valor por defecto **`SIG_DFL`** **(Nota**: en la implementación de señales POSIX de SUN-Solaris cuando una señal es esperada el manejador se desactiva por defecto).

Dentro de la especificación POSIX existen varias funciones para bloquear un proceso o thread esperando una señal. La más antigua recogida en la especificación POSIX 1 es la función ***`sigsuspend()`***, diseñada para trabajar con procesos, aunque también puede ser utilizada en programas con múltiples threads. Esta función tiene una limitación importante cuando el proceso o thread está esperando por varias señales, y es que no especifica que señal fue depositada. En a especificación POSIX 1.b se añadió una nueva función ***`sigwait()`*** que es la recomendada para aplicaciones multithreads y que añade la funcionalidad de devolver la señal que fue depositada. Se recomienda utilizar siempre esta última función.

```
int sigwait(sigset_t *set); // Multithread

int sigwait(const sigset_t *set, int *sig);
(-D_POSIX_PTHREAD_SEMANTICS)

int sigwaitinfo(const sigset_t *set, siginfo_t *info);

int sigtimedwait(const sigset_t *set, siginfo_t *info,
           const struct timespec *timeout);
```


Si se trabaja con señales de tiempo real (pueden ser encoladas con un mecanismo de prioridades), se dispone de las funciones ***`sigwaitinfo()`*** y ***`sigtimedwait()`***. Incorporan un parámetro adicional (***`info`***) donde se almacena la información extendida disponible en este tipo de señales (ver apartado 13.4.2). La última función permite especificar un tiempo de espera máximo (timeout), si se sobrepasa dicho intervalo de tiempo ni no se deposita ninguna señal la función devuelve el control.

13.6.1 Espera de señales mediante sigwait()

La función ***sigwait()***proporciona información de la señal depositada. Se pueden encontrar dos prototipos según la librería utilizada. El primero más habitual tiene un parámetro que es un **conjunto de señales** de tipo sigset_t (máscara) en la que debemos indicar aquellas señales que pueden despertarlo (un proceso puede estar atendiendo varias señales). La señal depositada es indicada en el valor devuelto por la función. El segundo prototipo precisa compilar la aplicación con el modificador `-D_POSIX_PTHREAD_SEMANTICS`, e incluye un parámetro adicional ***`int *sig`***, utilizado para devolver la señal que ha sido depositada, siguiendo el nuevo formato POSIX (se recomienda que los valores devueltos por una función indiquen solo el estado de error, utilizando parámetros para devolver cualquier otra información). En todo caso, la funcionalidad es la misma.

La función queda suspendida hasta que una señal del conjunto esté pendiente, devolviendo como resultado la señal depositada. Realiza tres operaciones de forma atómica:

1. Desbloquea el conjunto pasado como parámetro en la máscara de bloqueo del proceso o thread. Para evitar la pérdida de eventos las señales deberán estar bloqueadas previamente a la espera de la señal.
2. Queda a al espera de que se deposite alguna señal que no esté bloqueada
3. Cuando es depositada la señal, se reestablecen los bloqueos anteriores y se devuelve la señal depositada.

13.6.2 Espera de señales de tiempo real: sigwaitinfo(), sigtimedwait()

La función ***`sigwaitinfo()`*** permite la espera de señales de tiempo real que pueden encolarse de forma ordenada por prioridades. Se utiliza con señales en el rango [`SIGRTMIN-SIG_RTMAX`]. Cuanto menor es el valor de la señal mayor es la prioridad y antes será depositada

Incorpora, asimismo, la posibilidad de utilizar información adicional a través de la estructura ***`sig_info`***. En ella se incluye el número de la señal, la causa de la señal, el valor enviado, etc.

El proceso de bloqueo de la señal y espera es similar a las funciones vistas anteriormente, con la única salvedad de que la función ***`sigtimedwait()`*** permite especificar un intervalo máximo de bloqueo mediante el parámetro **`timeout`** de tipo **`timespec`** (ver tema 9). En las aplicaciones de tiempo real es importante evitar bloqueos indefinidos de forma que exista en las tareas un tiempo de respuesta máximo.

Un problema que debemos resolver (con ambas funciones) es ***¿qué ocurre si la señal se deposita antes de que se ejecute la llamada a `sigwaitinfo()` o `sigwait()`?***. En este caso se ejecutaría el manejador pero la señal pasaría inadvertida para el thread, que al ejecutar la función de espera de la señal quedaría bloqueada. Se perdería de este modo uno o varios eventos. Para resolverlo debemos mantener bloqueadas todas las señales en todo el proceso desde el comienzo de su ejecución. De este modo nos aseguramos que no se deposite ninguna señal antes de que llamemos a `sigwait()`. La propia llamada a `sigwait()` va a desbloquear de forma *atómica* las señales indicadas en el conjunto pasado como parámetro, con lo que aseguramos que no perderemos ningún evento.

Como se indicó anteriormente, una buena norma es bloquear todas las señales que nos interesen en la configuración inicial (main) antes de crear ningún thread, de este modo aseguramos que los thread creados posteriormente hereden las señales bloqueadas.

A continuación se presenta un ejemplo completo de manejo de señales para la ejecución asíncrona de tareas.


```
#define _REENTRANT
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <time.h>
#include <signal.h>

#define SIGNUM          SIGRTMAX

timespec_t  RETARDO ={ 10, 0L};     // retardo de 10s
void * ThreadSig(void *arg);
void MostraDatosSig( siginfo_t *info);


/*******************************
      función main
 *******************************/
int main(int argc, char *argv[])
{
      sigset_t sigset;        // conjunto de señales
      struct sigaction act;   // manejador señal
      pthread_t th_id;        // identificador para el thread
      pthread_attr_t attr;    // atributos de los threads
      union sigval val;       // paso de valor a una señal

      // bloquea la señal:
      // los threads creados posteriormente heredan la máscara
      sigemptyset(&sigset);
      sigaddset(&sigset, SIGNUM);
      pthread_sigmask(SIG_BLOCK, &sigset, NULL);
      printf("Thread main: Señal # %d bloqueada por el proceso: %d\n",
                  SIGNUM, getpid());


      printf("Thread main: Crea el thread de señal y espero\n\n");
      pthread_attr_init (&attr);
      pthread_create (&th_id, &attr, ThreadSig, (void *)SIGNUM);


      nanosleep (&RETARDO, NULL); // espera un rato

      // desde la consola se puede enviar una señal al proceso con
      //    kill -s <SIGNUM> <PID>

      printf("\nThread main: envío una señal al proceso y  espero\n");
      val.sival_int=1;  // valor pasado al manejador de la señal
      sigqueue( getpid(), SIGNUM,  val);

      nanosleep (&RETARDO, NULL);  // espera un rato
      // termina el thread de señal
      if (pthread_cancel(th_id)!=0)
            perror("Error cancelando thread");
      else
            printf("Thread main: Thread de señal # %d, ha sido cancelado\n",
                                                            th_id);

      printf("Thread main, # %d termina\n", pthread_self());
      pthread_exit(NULL);
}
```

```
/*********************************************************
 Thread de gestión de la señal
 *********************************************************/
void * ThreadSig(void *arg)
{
      int sig;
      sigset_t sigset;  // conjunto de señales
      siginfo_t   info; // información de la señal RT

      sig = (int)arg;

      // permite la cancelación del thread
      pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, NULL);
      pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);

      printf("Thread señal #%d: espero la señal # %d\n",
                                  pthread_self(), sig);

      // conjunto de señales que vamos a esperar
      sigemptyset(&sigset);
      sigaddset(&sigset, SIGNUM);

      // Espera la cancelación o la señal
      while(1)
      {
            // desbloquea y espera la señal de forma atómica
            sig=sigwaitinfo(&sigset, &info);
            // sigwaitinfo es un punto de cancelación

            if (sig!=-1)
            {     printf("----- Señal %d depositada en el thread # %d\n",
                                          sig, pthread_self());
                  MostraDatosSig(&info);
            }
            sched_yield();  // libera la CPU  para otro Thread
      }
      pthread_exit(NULL);
}


/*********************************************************
 Muestra datos de la señal siginfo_t
 *********************************************************/
void MostraDatosSig( siginfo_t *info)
{
      printf("info. Señal # %d,  Valor: %d ",
            info->si_signo, info->si_value.sival_int);

      printf("Code: (%d) ",info->si_code);
      if( info->si_code == SI_USER)
            printf("SI_USER \n" );
      else if( info->si_code == SI_TIMER )
            printf("SI_TIMER \n" );
      else if( info->si_code == SI_QUEUE )
            printf("SI_QUEUE \n" );
      else if( info->si_code == SI_ASYNCIO )
            printf("SI_ASYNCIO \n" );
      else if( info->si_code == SI_MESGQ )
            printf("SI_MESGQ \n" );
}
```


13.8 Resumen de funciones POSIX de manejo de señales

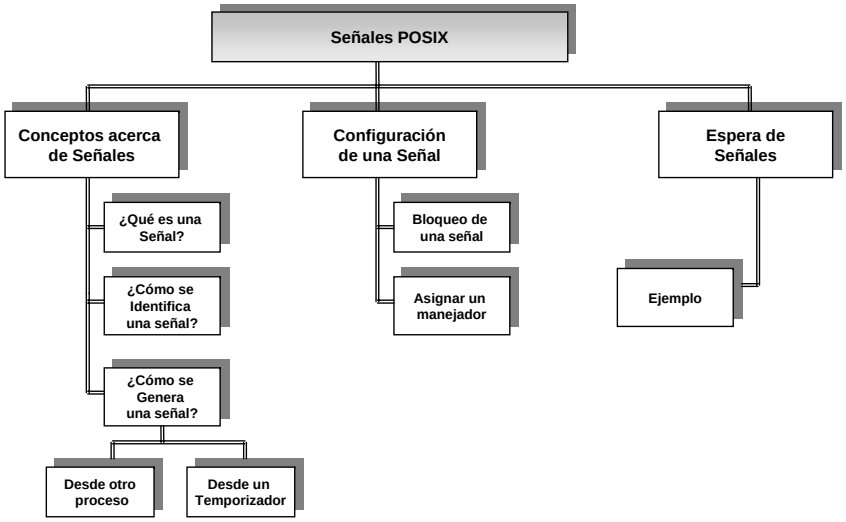
A continuación se resumen las estructuras de datos y funciones POSIX estudiadas que permiten el manejo de señales:

TIPO DE DATOS	CABECERA	DESCRIPCIÓN
`Union sigval { int sival_int; void *sival_ptr; };`	<signal.h> <sys/types.h>	Especifica el valor devuelto al manejador de la señal
`sigset_t`		Conjunto de señales
`struct sigaction { int sa_flags; union { void (*_handler)(int); void (*_sigaction)(int, siginfo_t *, void *); } _funcptr; sigset_t sa_mask; int sa_resv[2]; }; #define sa_handler _funcptr._handler #define sa_sigaction _funcptr._sigaction`		Estructura de configuración del manejador de la señal
`struct timespec { time_t tv_sec; long tv_nsec; };`	<time.h>	Tiempo en nanosegundos

FUNCIÓN	CABECERA	DESCRIPCIÓN
`int kill(pid_t pid, int sig);`	`<signal.h>`	Envía una señal a un proceso
`int sigsend(idtype_t idtype, id_t id, int sig);`		Envía una señal a un conjunto de procesos
`int sigqueue(pid_t pid, int signo, const union sigval value);`		Envía una señal a un proceso junto con un valor al manejador. La señal es encolable
`int sigemptyset(sigset_t *set);`	`<signal.h>`	Configura un conjunto de señales vacío (todas desactivas)
`int sigfillset(sigset_t *set);`		Configura un conjunto de señales con todas activadas
`int sigaddset(sigset_t *set, int signo);`		Añade una señal a un conjunto (activa)
`int sigdelset(sigset_t *set, int signo);`		Borra una señal de un conjunto (desactiva)
`int sigprocmask(int how, const sigset_t *set, sigset_t *oset);`	`<signal.h>`	Configura la mascara de bloqueo
`int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset);`		Configura la mascara de bloqueo
`int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);`		Configura el manejador de una señal
`int sigsuspend(const sigset_t *set);`	`<signal.h>`	Suspende un proceso o thread hasta que sea depositada una señal
`int sigwait(sigset_t *set);`		Suspende un proceso o thread hasta que sea depositada una señal. Devuelve el número de la señal.
`int sigwait(const sigset_t *set, int *sig); (-D_POSIX_PTHREAD_SEMANTICS)`		
`int sigwaitinfo(const sigset_t *set, siginfo_t *info);`		Suspende un proceso o thread hasta que sea depositada una señal de tiempo real. Devuelve una estructura con información adicional.
`int sigtimedwait(const sigset_t *set, siginfo_t *info, const struct timespec *timeout);`		Suspende un proceso o thread hasta que sea depositada una señal de tiempo real. Se especifica un intervalo de espera máximo. Devuelve una estructura con información adicional.

Resumen del capítulo

Este capítulo se ha dedicado a ofrecer una visión general del manejo de señales orientado fundamentalmente a al ejecución de tareas ante eventos asíncronos. El siguiente cuadro resume de forma esquematizada el contenido del capítulo



Bibliografía del capítulo

"Real-Time Systems and Programming Languages"(2ª edición)
A. Burns, A. Wellings, Addison-Wesley 1997

"Sistemas Operativos. Conceptos Fundamentales" (3ª ed.)
A. Silberschatz, J. Peterson, P. Galvin, Ed. Addison-Wesley, 1992

"UNIX Programación Práctica"
K.A. Robbins, S. Robbins, Prentice-Hall 1997

"Multithreading Programming with Pthreads"
B. Lewis, D.J. Berg, Prentice-Hall Sun Microsystems, 1998

"Programming for the Real World. POSIX.4"
B. O. Gallmeister, Ed. O'Reilly & Associates 1995