Señales

Ing. Eddie Angel Sobrado Malpartida

Que es una señal?

- Una **señal** es una notificación (mensaje) por software (interrupciones software: suceso asíncrono) enviadas a uno o mas proceso/thread para informarle de la ocurrencia de un evento. Este mensaje no es más que un número entero.

Similar a las **interrupciones** 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 un fragmento de código situado fuera del flujo normal de ejecución.

Señales en POSIX

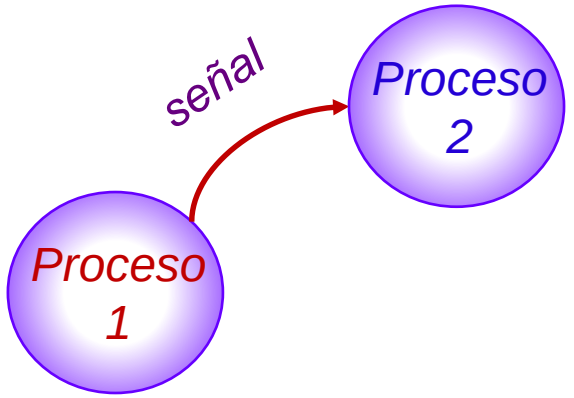
- En el caso de una **señal**, la aparición del evento es gestionada por el S.O. decidiendo cuando se interrumpe el proceso o thread que tiene actualmente los recursos, para activar la tarea asociada a la señal.

SEÑAL es a proceso como
INTERRUPCIÓN es a procesador

El origen de una señal puede ser de un PROCESO o del SISTEMA OPERATIVO.

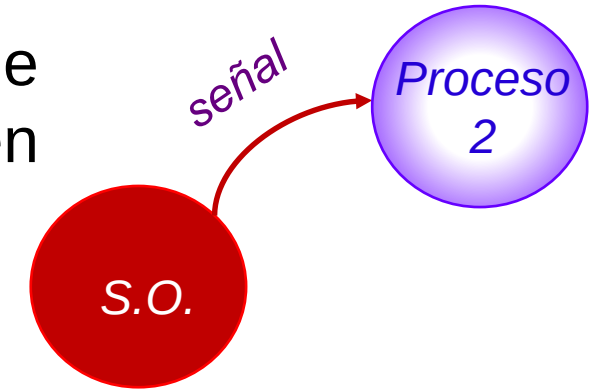
Origen de una señal: Proceso a Proceso

- Un proceso puede enviar una señal a otro proceso que tenga el mismo identificador de usuario (UID), pero no a los que lo tengan distinto
- Un proceso del superusuario puede mandar una señal a cualquier proceso, con independencia de su UID (‘root’ a cualquier UID o GID)
- Un proceso también puede mandar una señal a un grupo de procesos, que han de tener su mismo UID.



Origen de una señal: Proceso a Proceso

El sistema operativo también toma la decisión de enviar señales a los procesos cuando ocurren determinadas condiciones.

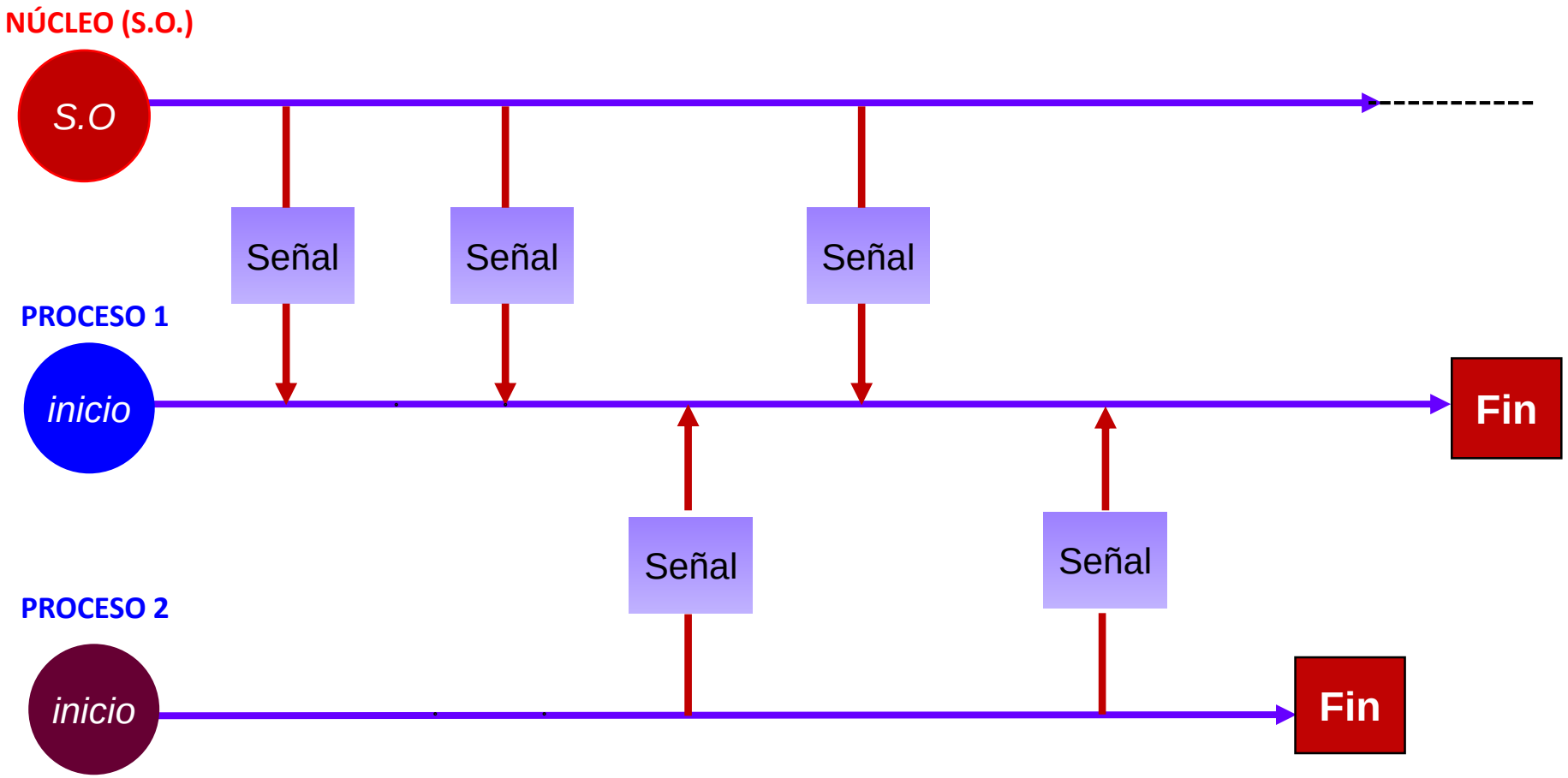


Por ejemplo, las excepciones de ejecución programa:

- ✓ el desbordamiento en las operaciones aritméticas,
- ✓ la división por cero,
- ✓ el intento de ejecutar una instrucción con código de operación incorrecto o de direccionar una posición de memoria prohibida

las convierte el sistema operativo en señales al proceso que ha causado la **excepción**

Origen de una señal



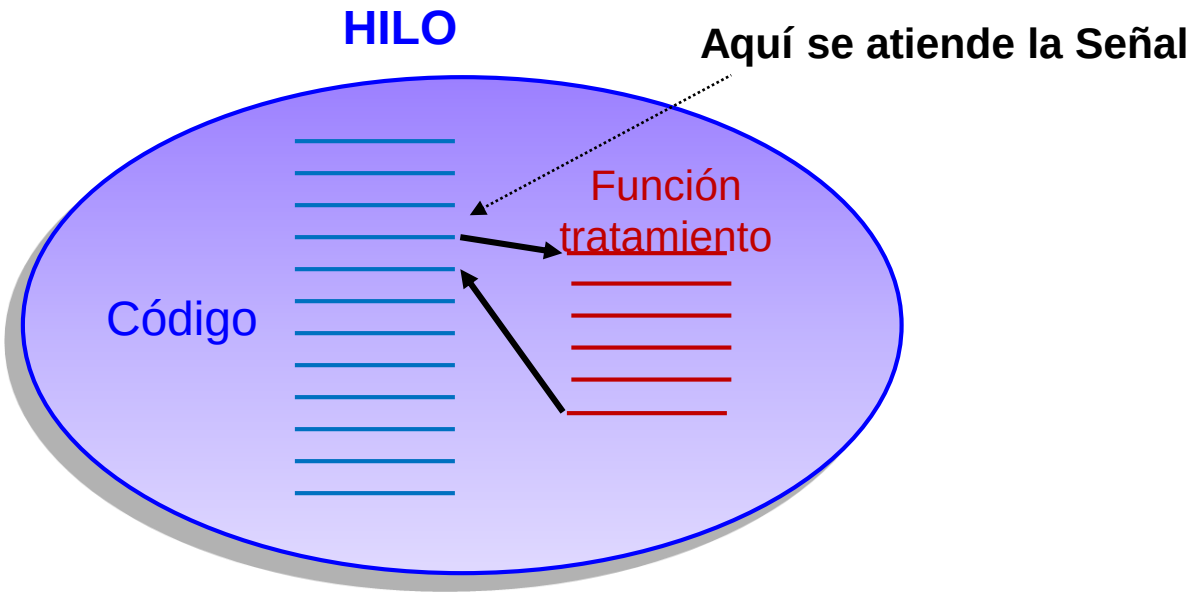
Comportamiento del proceso que recibe una señal

- Cuando un proceso recibe una señal puede:
  - ✓ Ignorar a la señal, cuando es inmune a la misma
  - ✓ Invocar la rutina de tratamiento por defecto
  - ✓ Invocar a una rutina de tratamiento propia

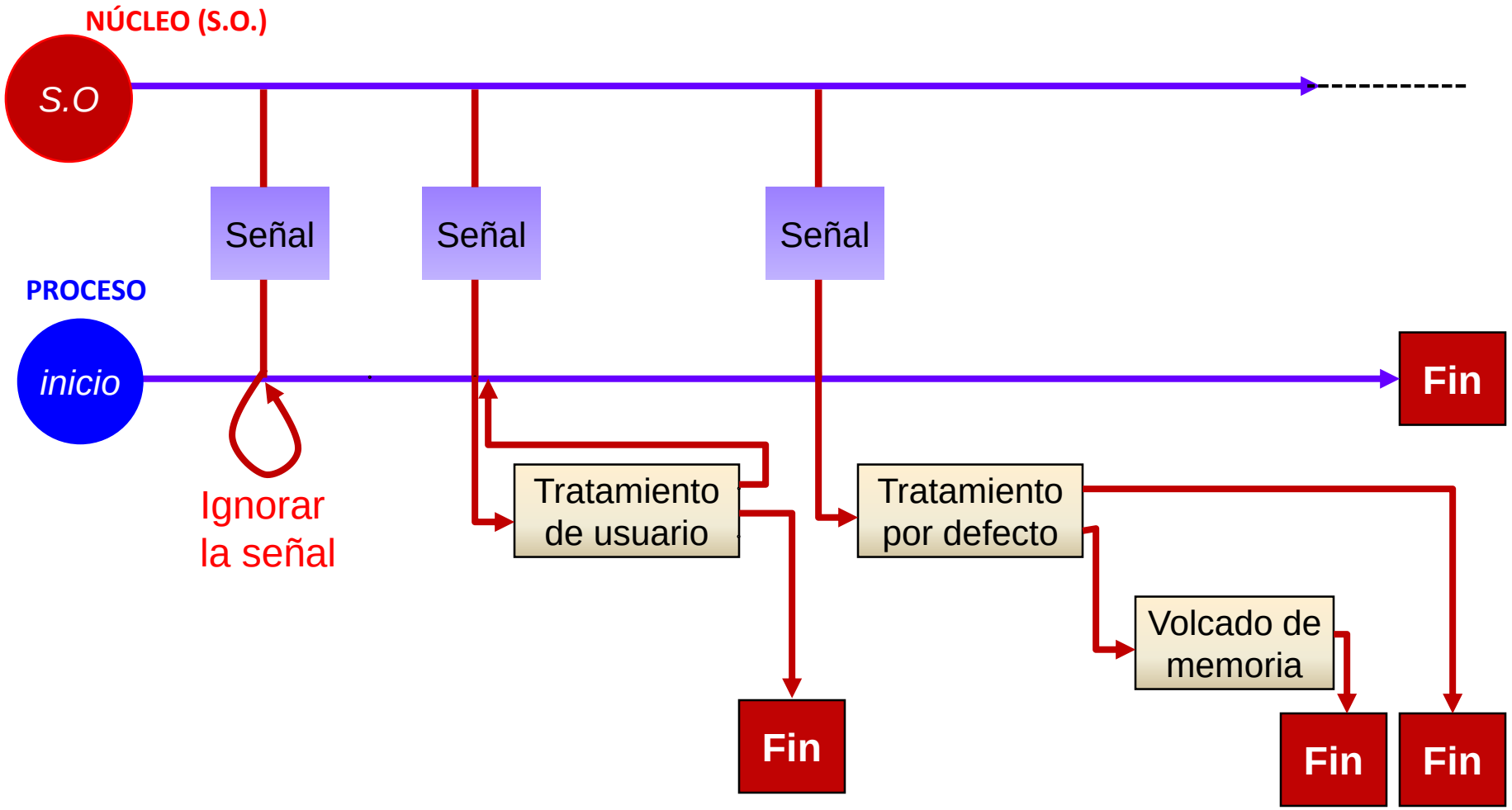
Invocando a una rutina de tratamiento

Rutina de tratamiento

1. El proceso detiene su ejecución en la instrucción de máquina que está ejecutando.
2. Bifurca a ejecutar una rutina de tratamiento de la señal, cuyo código ha de formar parte del propio proceso.
3. Una vez ejecutada la rutina de tratamiento, sigue la ejecución del proceso en la instrucción en el que fue interrumpido.



Comportamiento del proceso que recibe una señal



Estados de una señal

Estados por la que pasa una señal

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

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

Estados por la que pasa una señal

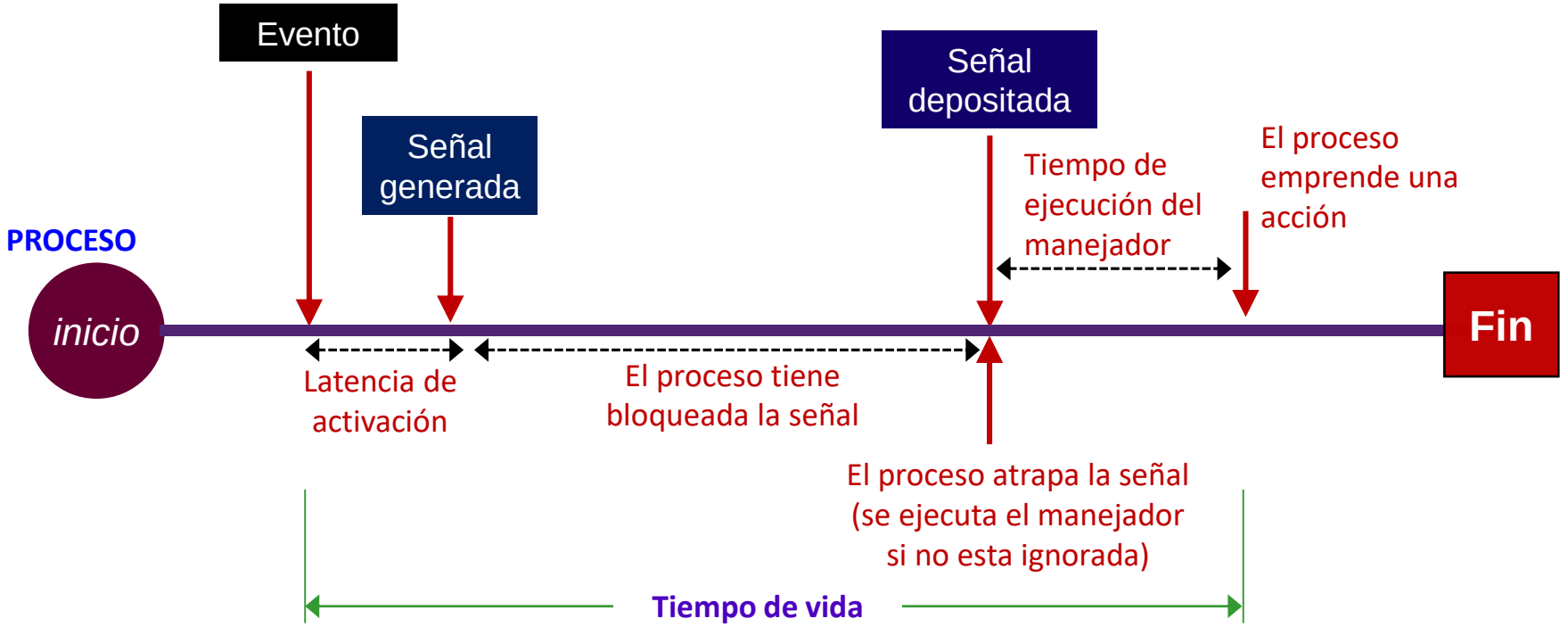
4. Pendiente: Se dice que una señal está pendiente si ha sido generada pero todavía no está depositada.

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

6. Ignorada: 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.

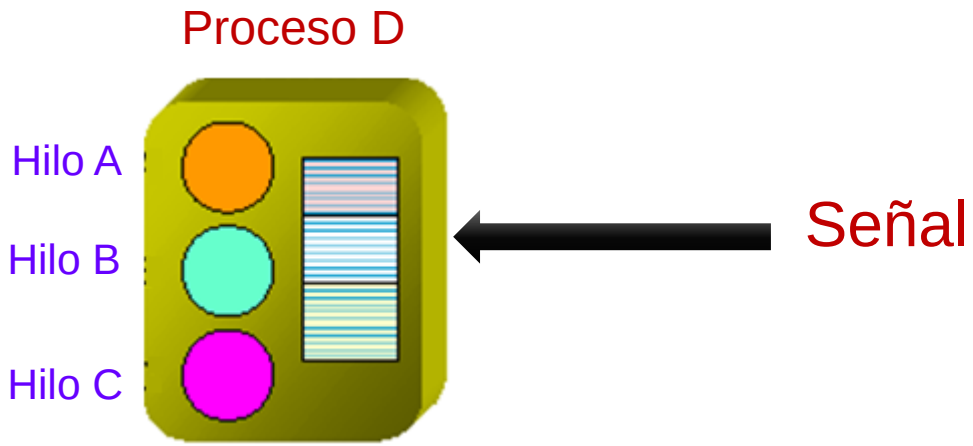
Estados por la que pasa una señal

7. Bloqueada: 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.



Tipos de señales

- 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.



Tipos de señales

- 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

Tipos de señales

2. Señales asíncronas: producidas por llamadas explícitas (kill) o por eventos no asociados al código en ejecución. Aquí la señal va dirigida a todos los threads del proceso. Pero una señal solo puede ser depositada por un thread

¿Cuál será el thread que deposite la señal?.

Pues bien, la recibirá uno cualquiera que cumpla una de las siguientes condiciones:

- ✓ El thread está bloqueado 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)

Tipos de señales

- 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

Tipos de señales

- Las señales generadas por un error de un *thread (suceso síncrono)* se envían sólo a ese *thread*
- Las generadas por un suceso asíncrono (p. ej. E/ S) se envían al proceso
- Las generadas por programa se pueden enviar a un *thread* o a todo un proceso

Identificadores de Señales

Identificadores de señal

- Cada tipo de señal se identifica mediante un número entero positivo para facilitar su utilización todas las señales tienen asociado un nombre simbólico que comienza con el prefijo SIG
- Se definen constantes (etiquetas) para dar nombres simbólicos a las señales

Identificadores de señal

Listado de Señales (Sun-Solaris)

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 Child status change
/SIGCHLD Child status change alias (POSIX)
SIGPWR power-fail restart
SIGWINCH window size change
SIGURG urgent socket condition
SIGPOLL / pollable event occurred
SIGIO 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

Identificadores de señal

- **SIGABRT** : *‘abort signal’*
  - ✓ Terminación anormal
  - ✓ Se genera cuando se llama a la función abort()
  - ✓ Causa la terminación del proceso
  - ✓ No debe ser ignorada
- **SIGALRM** : *‘alarm signal’*
  - ✓ Fin de temporización
  - ✓ Se genera cuando expira el tiempo programado con alarm() o con setitimer()
- **SIGFPE** : *‘floating point exception signal’*
  - ✓ operación aritmética errónea
  - ✓ Ejemplo: dividir por cero, desbordamiento (overflow)

Identificadores de señal

- **SIGINT** : *‘interruption signal’*
  - ✓ Señal de atención interactiva
  - ✓ Se envía a proceso asociado a un terminal (teclado) cuando se pulsa la tecla interrupción [CTRL-C]
  - ✓ Acción por defecto terminar el proceso
- **SIGKILL**: *‘kill signal’*
  - ✓ Señal de terminación abrupta
  - ✓ No puede ser ignorada ni armada
  - ✓ Acción por defecto genera ‘core’ + terminar(matar) proceso

Identificadores de señal

- **SIGSEV** : *‘segment violation signal’*
  - ✓ Referencia a memoria inválida
  - ✓ Acción por defecto genera ‘core’ + terminar proceso
- **SIGTERM**: *‘termination signal’*
  - ✓ Terminación, por ejemplo, ante ‘shutdown’ del sistema
  - ✓ Similar SIGKILL, pero puede ser ignorada y armada
  - ✓ Acción por defecto terminar el proceso
  - ✓ Finalización controlada. Se envía para indicarle a un proceso que debe acabar su ejecución. Puede ser ignorada. Definiremos una rutina para definir el comportamiento del sistema si recibe esta señal.
- **SIGUSR1 & SIGUSR2:** *‘user signal’*
  - ✓ Reservadas para uso del programador
  - ✓ Ninguna aplicación estándar va a usarla y su significado es el que quiera definir el programador en su aplicación
  - ✓ Acción por defecto terminar el proceso
- **SIGCLD**
  - ✓ Terminación de algún proceso hijo. Se envía al proceso padre. Ignorada por defecto

Señal generados por el SO

- 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.

Señal generados por el usuario

- En la tabla se observan:
  - ✓ Las señales definibles por el usuario **SIGUSR1**, **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.

Prioridades de las señales

- Todas las señales tienen la misma prioridad. A diferencia de las interrupciones hardware, las señales se procesan siguiendo la filosofía FIFO (First In First Out).
- Cuando una señal se envía a un proceso, se ejecuta el manejador correspondiente, y mientras este manejador no termine su ejecución, ninguna otra señal podrá ser recibida por el proceso anterior.

Asignación de un manejador de señal

- Puede ocurrir que un proceso no quiera ejecutar el manejador por defecto asociado a un tipo de señal.
- Para este caso, existe una función *signal()* que especifica qué tratamiento debe realizar un proceso al recibir una señal.

**signal (int sig, void (*action)())*

donde:

- sig: Número entero que representa una señal definida en /usr/include/signal.h
- *action: Especifica la dirección de un manejador de señal, dada por el usuario en el proceso. Es la acción que se tomará al recibir la señal y puede tomar tres clases de valores.

Asignación de un manejador de señal

- Existen dos manejadores predefinidos en /usr/include/signal.h:

  SIG_DFL: Manejador por defecto. *Por lo general, es la de abortar e incluso dar core.*

  SIG_IGN: Manejador que ignora la señal recibida.

- Para las señales SIGKILL, SIGSTOP y SIGCONT no es posible asignar un manejador que no sea el de defecto.

Como se genera una señal

- Una señal se puede *producir* o *generar* de varias formas, por ejemplo:
  - ✓ fallos de hardware
  - ✓ expiración de temporizadores
  - ✓ un acción explícita de otro proceso o *thread,* invocando la función *kill()* u otra similar

Como 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.

Como se genera una señal

- 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.
- Veremos dos métodos de generación de señales.

1.Generación una señal desde otro proceso

- Para poder enviar una señal a otro proceso o grupo de procesos, es necesario realizar la llamada al sistema *kill().*

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

donde:

- pid: Identificador del proceso al cual va dirigida la señal.
- sig: identifica la señal a mandar.
- ret: 0 (Éxito) 1 (Error)

Si pid > 0, es el PID del proceso al que enviamos la señal
Si pid = 0, la señal es enviada a todos procesos del grupo
Si pid = -1, la señal es enviada a los procesos cuyo id. Real es igual al id. efectivo del proceso que la envía.

1.Generación una señal desde otro proceso

- La llamada al sistema pause(), provoca la suspensión de la ejecución del proceso, hasta que se recibe una señal. Siempre retorna -1.

- Otra forma sencilla de enviar una señal a un proceso es utilizar el comando

kill -signal pid desde la consola.

Generación y entrega

- **Armado de una señal:** Indicarle al sistema operativo el nombre de la rutina que ha de tratar cuando llegue ese tipo de señal
  - ✓ En POSIX con la función *sigaction( )*
  - ✓ En System V con la funcion *signal( )*

sigaction y *signal* permiten instalar manejadores de señales

Generación y entrega

- Una señal se genera cuando ***ocurre*** el suceso que la produce
- Una señal se entrega cuando se ***produce*** la acción asociada en el proceso o *thread*
- Un proceso cuando recibe una señal puede optar por tres posibles alternativas para procesarla:
  - ✓ *Bloquear* Temporalmente
  - ✓ *Ignorar* la señal recibida
  - ✓ *Ejecutar* la *acción por defecto* asociada a la señal (normalmente terminar el proceso, la aporta el kernel)
  - ✓ *Ejecutar (Manejar)* la señal mediante una función definida por el programador en el propio proceso