Señales POSIX

- Introducción
- Versión original: Señales POSIX 1003.1a
- Señales POSIX 1003.1b (señales “de tiempo real”)
- Tratamiento de señales en procesos multihilo
- Inconvenientes y ventajas de las señales

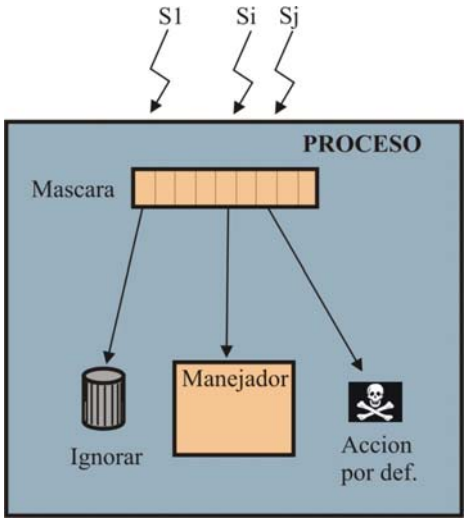
Señales POSIX

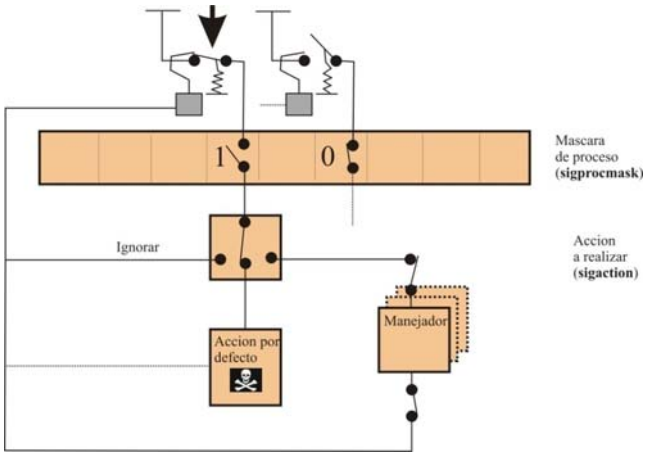
- Introducción
  - Aparecen con UNIX como medio para notificar eventos
  - Normalizadas en POSIX 1003.1a
  - Mejoradas en POSIX 1003.1b (“señales de tiempo real”
  - Utilidad: Notificación de eventos de diversos servicios POSIX, en particular los temporizadores

Versión original: Señales POSIX 1003.1a

- Normalización de señales UNIX
- En recepción, dos problemas a considerar:
  - Máscara de señales del proceso (las señales enmascaradas quedan pendientes)
  - Acción a realizar:
    - Acción por defecto (habitualmente: abortar el proceso)
    - Ignorar
    - Ejecutar manejador (código excepcional que interrumpe temporalmente el proceso)
- Algunas señales:
  - SIGTERM, SIGKILL, SIGINT, SIGQUIT
  - SIGFPE
  - SIGALRM
  - SIGUSR1, SIGUSR2

Señales (1003.1a)





Versión original: Señales POSIX 1003.1a

- Tratamiento asíncrono:
  - Es el mecanismo previsto originalmente
  - Se programa un manejador como acción a realizar
  - Al recibirse la señal, el manejador se ejecuta, interrumpiendo el proceso temporalmente
  - Posibles problemas con llamadas no “async-safe”
  - Interrumpe las esperas (error EINTR)
- Tratamiento síncrono:
  - El proceso espera a la llegada de la señal
  - No está previsto: Puede hacerse utilizando la interrupción de esperas por error EINTR
  - Siempre ha de ejecutarse el manejador
- Enviar señales:

  int **kill**(pid_t destino, int señal);

Señales POSIX 1003.1a:Tratamiento asíncrono

Ejemplo: Manejador para SIGTERM y SIGINT

```
(…)
#include <signal.h>
(…)
void manej(int s) { …. }
(…)
struct sigaction acc;
sigset_t sen;

/* Programar manejador */
acc.sa_flags = 0;
acc.sa_handler = manej;
sigemptyset(&acc.sa_mask);
sigaction(SIGTERM, &acc, NULL);
sigaction(SIGINT, &acc, NULL);
```

```
/* Desenmascarar ambas señales */
/*Antes hay que hacer un conjunto */
sigemptyset(&sen);
sigaddset(&sen, SIGTERM);
sigaddset(&sen, SIGINT);
sigprocmask(SIG_UNBLOCK, &sen, NULL);
/* A partir de ahora se recibirán señales y
   entrará manej  asíncronamente */
(…)
```

Señales POSIX 1003.1a: Tratamiento síncrono

Ejemplo: Esperar SIGUSR1

```
(…)
#include <signal.h>
(…)
void manej(int s) { …. }
(…)
struct sigaction acc;
sigset_t sen;
int res;

/* Programar manejador */
acc.sa_flags = 0;
acc.sa_handler = manej;
sigemptyset(&acc.sa_mask);
sigaction(SIGUSR1, &acc, NULL);

/* Desenmascarar SIGUSR1 */
sigemptyset(&sen);
sigaddset(&sen, SIGUSR1);
sigprocmask(SIG_UNBLOCK, &sen, NULL);
/* Es necesario que entre manej para romper
     la espera */
(…)
/* Esperar SIGUSR1 */
res = sleep(3600);
if(res != 0) {
     printf("Ha llegado SIGUSR1\n");
     hacer_cuando_SIGUSR1(…);
} else {
     printf("Error: No llega SIGUSR1!!\n");
     hacer_trat_error(…);
}
(…)
```

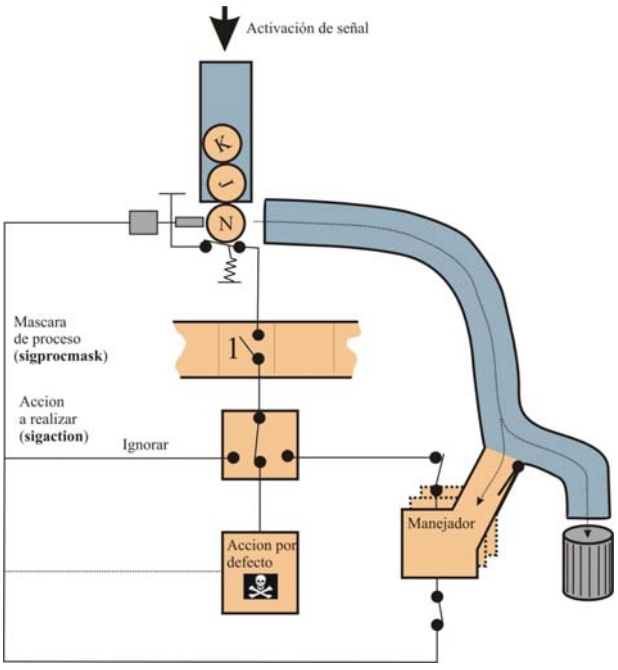

Ejemplo: Sobretiempo con señales

```
#include <unistd.h>
#include <signal.h>
int timout;   /* Flag de sobretiempo */
void maneja(int sign) {
     timout = 1;
}
 void main(void) {
     int fin;  /* Flag de fin */
     struct sigaction action;
     sigset_t set;

     /* Programación de la acción a realizar */
     action.flags = 0;
     action.sa_handler = maneja;
     sigemptyset(&action.sa_mask);
     sigaction(SIGALRM, &action, NULL);

     /* Desbloqueo de SIGALRM */
     sigemptyset(&set); sigaddset(&set,
     SIGALRM);
     sigprocmask(SIG_UNBLOCK, &set, NULL);

     /* Después se programa el "despertador" */
     timout = 0;
     alarm(20);
     fin = 0;

     /* Algoritmo con sobretiempo */
     while(!fin && !timout) {
        fin = funcion_iteracion(…);
     }

     /* Analizar lo que ha sucedido */
     if(fin) alarm(0); /* Ya no debe saltar */
     if(timout) {
          printf("Alarma! Sobretiempo!\n");
          < algoritmo alterrnativo>
     }
}
```

Señales POSIX 1003.1b (señales “de tiempo real”)

- Problemas de POSIX 1003.1a:
  - Pocas señales de propósito general
  - No se encolan (pueden perderse señales)
  - No se conoce el orden de recepción
  - No pueden transmitir datos adicionales
  - Son asíncronas (manejador imprescindible)
- Mejoras de POSIX 1003.1b:
  - Nuevas señales, de SIGRTMIN a SIGRTMAX
  - Encolamiento de señales pendientes
  - Dato adicional (entero o puntero)
  - Prioridad: Decreciente de SIGRTMIN a SIGRTMAX
  - Método de espera síncrona alternativo al manejador

Señales POSIX 1003.1b



Señales POSIX 1003.1b

- Para disponer de las nuevas funciones (encolamiento, dato, prioridad):
  - Programar el flag SA_SIGINFO (con **sigaction**)
  - Usar nuevo manejador (**sa_sigaction**) en lugar del antiguo (**sa_handler**)
  - Sólo llega dato si se ha enviado (imposible con **kill**): Nueva llamada **sigqueue**
- Tratamiento síncrono sin manejador:
  - Llamadas **sigwaitinfo** y **sigtimedwait**
  - Hay que **enmascarar** las señales que se esperan
  - Pueden desbloquearse por error EINTR por señales que **no se esperan** (solución: reintentar)
- Envío de señales:
  - Puede usarse **kill**, pero entonces no hay dato válido
  - Otra opción:

    int **sigqueue**(pid_t destino, int señal, union sigval dato);

Señales POSIX 1003.1b: Tratamiento asíncrono

Ejemplo: Manejador que recibe datos

```
(…)
#include <signal.h>
(…)
Int n = 0;
void mtr(int s, siginfo_t *e, void *p) {
    n = n + e->si_value.sival_int;
}
(…)
struct sigaction acc;
sigset_t sen;
```

```
/* Programar manejador */
acc.sa_flags = SA_SIGINFO;
acc.sa_sigaction = mtr;
sigemptyset(&acc.sa_mask);
sigaction(SIGRTMIN, &acc, NULL);

/* Desenmascarar SIGRTMIN */
sigemptyset(&sen);
sigaddset(&sen, SIGRTMIN);
sigprocmask(SIG_UNBLOCK, &sen, NULL);

/* A partir de ahora mtr  entrará
    asíncronamente y sumará en n los datos
    asociados a la señal */
(…)
```

Señales POSIX 1003.1b: Tratamiento síncrono

Ejemplo: Recepción de datos con **sigwaitinfo**

```
(…)
#include <signal.h>
(…)
Int n = 0;
void mtr(int s, siginfo_t *e, void *p) { }
(…)
struct sigaction acc;
sigset_t sen; siginfo_t info;

/* Programar manejador */
acc.sa_flags = SA_SIGINFO;
acc.sa_sigaction = mtr;
sigemptyset(&acc.sa_mask);
sigaction(SIGRTMIN, &acc, NULL);
```

```
/* Enmascarar SIGRTMIN */
/* Si no, no hay garantía de que funcione */

sigemptyset(&sen);
sigaddset(&sen, SIGRTMIN);
sigprocmask(SIG_BLOCK, &sen, NULL);

/* Bucle para sumar los datos */
/* mtr nunca entra */
while(1) {
    sigwaitinfo(&sen, &info);
    n = n + info.si_value.sival_int;
}
(…)
```


Señales POSIX 1003.1b: campo **si_code**

- Permite conocer la causa de la señal
- Opciones:
  - SI_QUEUE: Enviada por **sigqueue**; hay dato
  - SI_USER: Enviada por **kill**; no hay dato
  - SI_TIMER: Enviada por temporizador
  - Otras causas

```
(…)
sigset_t s; siginfo_t info;
/* Bloquear señal */
sigemptyset(&s);
sigaddset(&s, SIGRTMIN);
sigprocmask(SIG_BLOCK, &s, NULL);
(…)
/*Esperar señal; suponemos SA_SIGINFO
activado */
/* Si se usó kill para enviar el dato
   no es válido */
```

```
sigwaitinfo(&s, &info);
if(info.si_code == SI_QUEUE)
    printf("Dato: %d\n", info.si_value.sival_int);
else printf("No hay dato; enviada por kill\n");
else printf("Causa no esperada\n");
(…)
```

Señales en procesos multihilo

- Selección del hilo destinatario:
  - Tres casos:
    - Señal generada síncronamente por error: El hilo que la provoca
    - Señal enviada a un hilo con **pthread_kill**: Destinatario perfectamente definido
    - Señal enviada al proceso completo: Caso a discutir
  - Cada hilo tiene una máscara propia que se modifica con **pthread_sigmask**:
    - Tratamiento asíncrono (con manejador):
      - El manejador es común para todos los hilos
      - Si la señal está enmascarada en todos menos uno, el manejador se ejecuta en el contexto de ese hilo
      - En caso contrario, no se puede predecir el destinatario
    - Tratamiento síncrono (**sigwaitinfo** y otras):
      - Señal enmascarada en todos los hilos
      - Sólo un hilo esperando a la vez la señal, si no, dest. indefinido

Señales en procesos multihilo

- Particularidades de las funciones **pthread_xx**:
  - No se exige que sean “async-safe”: Posibles problemas para usarlas en un manejador
  - No se rompen las esperas cuando salta un manejador (no se genera el error EINTR)
- Algunas conclusiones:
  - Parece aconsejable en muchos casos tratar las señales síncronamente, dedicando hilos a esperar señales
  - Si se enmascaran las señales en **main**, el resto de los hilos heredará la máscara

Señales: Ventajas e inconvenientes

- Inconvenientes:
  - Tratamiento poco eficiente (sobre todo si es asíncrono)
  - Naturaleza asíncrona (salvo señales de 1003.1b)
  - Poco ancho de banda para transmitir datos
- Ventajas:
  - Son asíncronas (si esa funcionalidad es la que se necesita; con hilos, raramente)
  - Relativamente simples de programar
  - Orientadas a un proceso en particular (cualidad poco habitual)
  - No suelen verse afectadas por restricciones y se dispone siempre de ellas