dit
UPM

Sucesos asíncronos

Juan Antonio de la Puente
DIT/UPM

Sucesos asíncronos

- Un *suceso asíncrono* es una señal generada por un proceso que requiere atención por parte de otro proceso, independientemente de lo que esté haciendo
- Dos formas de continuar después de atender el suceso:
  - *reanudación*
    - » ej.: señales de POSIX
  - *terminación*
    - » ej.: transferencia de control asíncrona en Ada

Aplicaciones

- **Recuperación de errores**
  - cuando varios procesos colaboran la recuperación hay que hacerla conjuntamente
- **Cambios de modo**
  - debidos a cambios en el entrono o a averías
- **Cómputo impreciso**
  - se hace lo que se puede hasta que se agota el tiempo disponible
- **Interrupciones del operador**
  - a veces hay que dejar lo que se está haciendo y empezar de nuevo o cambiar de modo

Señales en POSIX.1

- Una señal representa un *suceso asíncrono* que afecta a uno o más procesos o *threads*

- 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

Identificadores de señal

- Cada tipo de señal se identifica mediante un número positivo
- Se definen constantes para dar nombres simbólicos a las señales

- Ejemplos:
  - SIGABRT — Terminación anormal
  - SIGALRM — Temporizador
  - SIGUSR1, SIGUSR2 — Definidas por el usuario

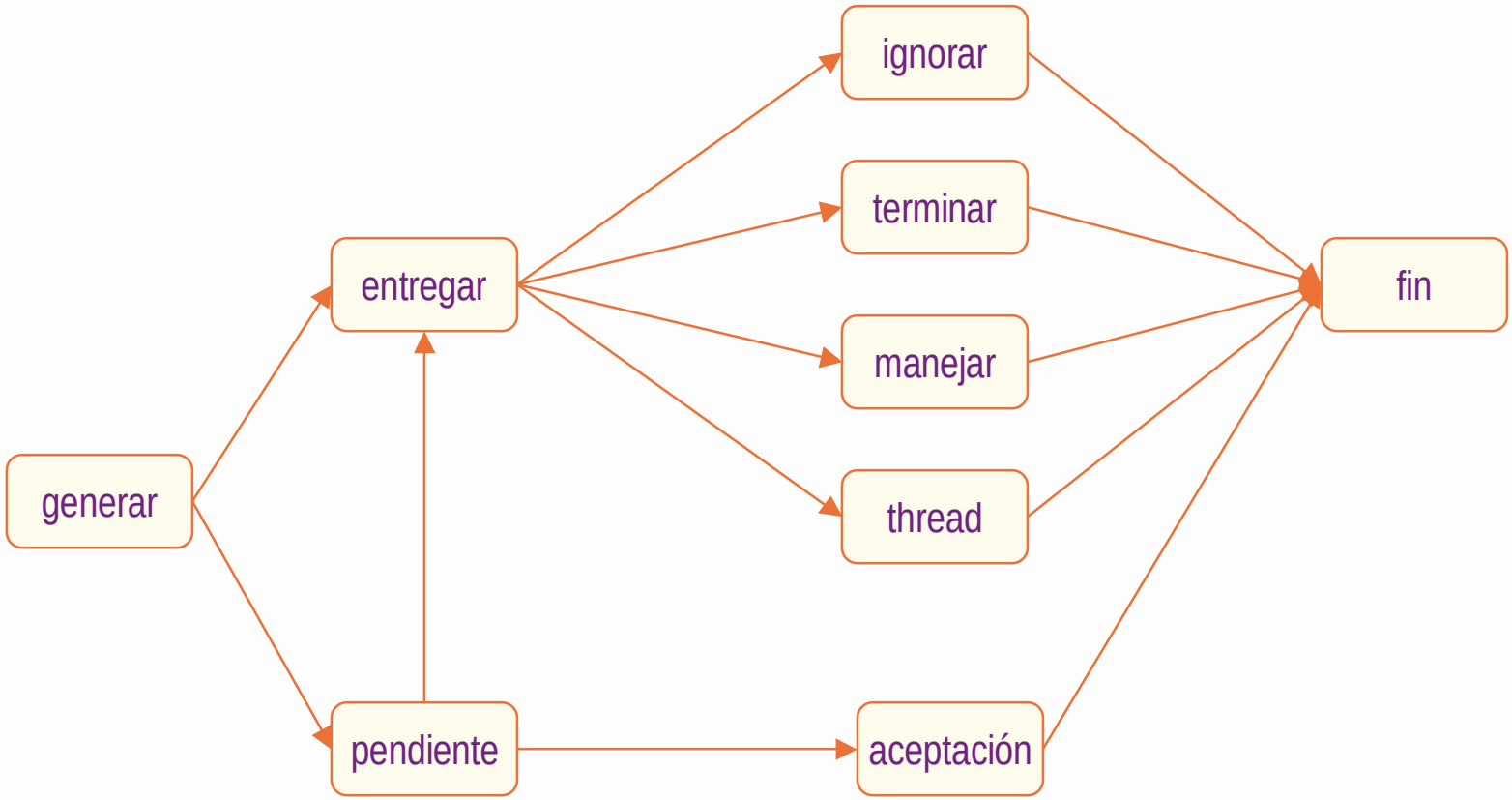
Generación y entrega

- 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*:
  - *Ignorar* la señal
  - Ejecutar la *acción por defecto* asociada a la señal (normalmente terminar el proceso)
  - *Manejar* la señal mediante una función definida por el programador

Bloqueo de señales

- Un proceso puede *enmascarar* o *bloquear* un tipo de señales
  - cada proceso o *thread* tiene una *máscara* que define las señales bloqueadas
  - la máscara se hereda inicialmente del padre
- Una señal bloqueada se *acepta* cuando se invoca una función `sigwait`
  - esta función se define en POSIX.1b

Esquema general



Señales de tiempo real (POSIX.1b)

- Tienen identificadores en el intervalo SIGRTMIN .. SIGRTMAX
  - como mínimo 32 señales
- Las señales de tiempo real tienen propiedades especiales:
  - se encolan
  - se aceptan en orden de prioridad (número de señal)
  - tienen un campo adicional de información
- Todas las señales se pueden aceptar de forma síncrona mediante la función `sigwait`
  - para ello deben estar enmascaradas (bloqueadas)

Señales y *threads*

- Las señales generadas por un error de un *thread* 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
- Una señal generada para un *thread* en el que está bloqueda se queda pendiente hasta que se acepta con *sigwait*
- Una señal generada para un proceso que está bloqueada en todos los *threads* se envía a un solo *thread* de los que hacen *sigwait* en la señal

Conjuntos de señales

```
/* signal.h */

typedef ... sigset_t;

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

Bloqueo de señales

```
/* signal.h */

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

- *how*: indica qué operación se hace:
  - SIG_BLOCK : se bloquean las señales de *set*
  - SIG_UNBLOCK: se desbloquean las señales de *set*
  - SIG_SETMASK: se hace la máscara igual a *set*
- *oset*: devuelve las señales que estaban bloqueadas antes
- *set* y *oset* pueden valer NULL

Envío y aceptación de señales

```
/* signal.h */

int kill          (pid_t pid,         int sig);
int pthread_kill (pthread_t thread, int sig);
```

```
/* signal.h */

int sigwait      (const sigset_t *set, int *sig);
int sigwaitinfo  (const sigset_t *set, siginfo_t *siginfo);
int sigtimedwait (const sigset_t *set, siginfo_t *siginfo,
   const struct timespec *timeout);
```

Ejemplo (1)

```
/*  thread que acepta SIGINT (^C) y escribe un mensaje */
#include <signal.h>
#include <pthread.h>

void wait_sigint (void)
{
  sigset_t set;
  int sig;
  int counter = 0;

  sigemptyset(&set);
  sigaddset(&set,SIGINT);
  pthread_sigmask(SIG_BLOCK, &set, NULL); /* bloquea SIGINT */
  /* todos los demás threads deben bloquear también SIGINT */
```

Ejemplo (2)

```
   if (sigwait(&set, &sig) != 0) {
     /* error en sigwait */
     pthread_exit( (void *)-1);
   }
   if (sig == SIGINT) {
     printf ("recibido SIGINT\n");
     pthread_exit(0);
   } else {
     /* señal inesperada */
     pthread_exit ((void *)-1);
   }
}
```

Manejadores de señales

```
/* signal.h */

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

int sigaction (int sig,
   const struct sigaction *reaction,
   struct sigaction *old_reaction);
```

- *sa_handler* puede ser SIG_DFL, SIG_IGN o un puntero a una función
- *sa_sigaction* se usa con señales de tiempo real

Transferencia de control asíncrona en Ada

- Es una forma especial de *select*:

```
select
  suceso;
  secuencia de instrucciones
then abort
  secuencia de instrucciones
end select;
```

- El suceso puede ser
  - Una llamada a una entrada

    ```
    Objeto.Entrada (...); -- puede ser una llamada a una tarea
    ```

  - Un retardo

    ```
    delay ...;
    ```