

Lección 3

Señales, excepciones y pipes

Sistemas Operativos
Ingeniería Informática

A recordar...

Antes de clase

Clase

Después de clase

Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:
las transparencias solo no son suficiente.
Preguntar dudas (especialmente tras estudio).

Ejercitar las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar **laboratorios** y **prácticas** de forma progresiva.

Lecturas recomendadas

Base



1. Carretero 2020:
 1. Cap. 5
2. Carretero 2007:
 1. Cap. 3.6 y 3.7
Cap. 3.9 y 3.13

Recomendada



1. Tanenbaum 2006:
 1. (es) Cap. 2.2
 2. (en) Cap.2.1.7
2. Stallings 2005:
 1. 4.1, 4.4, 4.5 y 4.6
3. Silberschatz 2006:
 1. 4

Contenidos

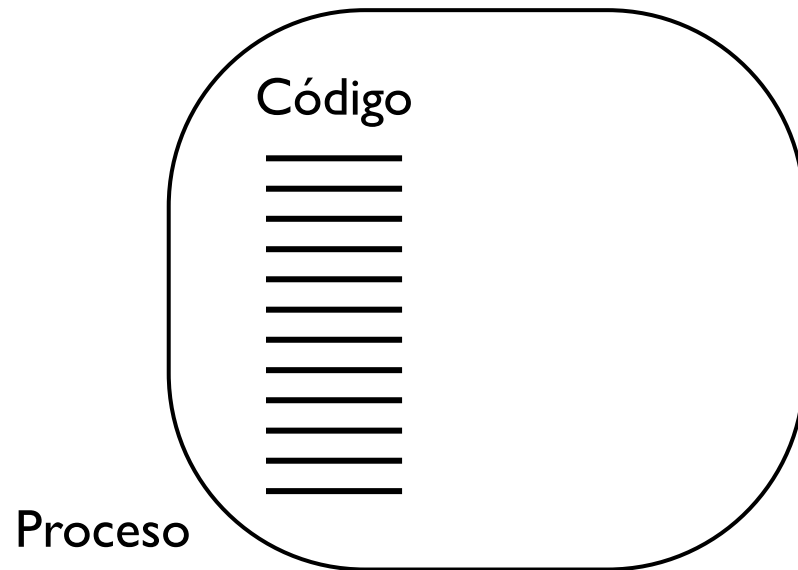
1. Señales y excepciones.
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (pipes).
 - ▶ Paso de mensajes local.

Contenidos

1. **Señales y excepciones.**
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (pipes).
 - ▶ Paso de mensajes local.

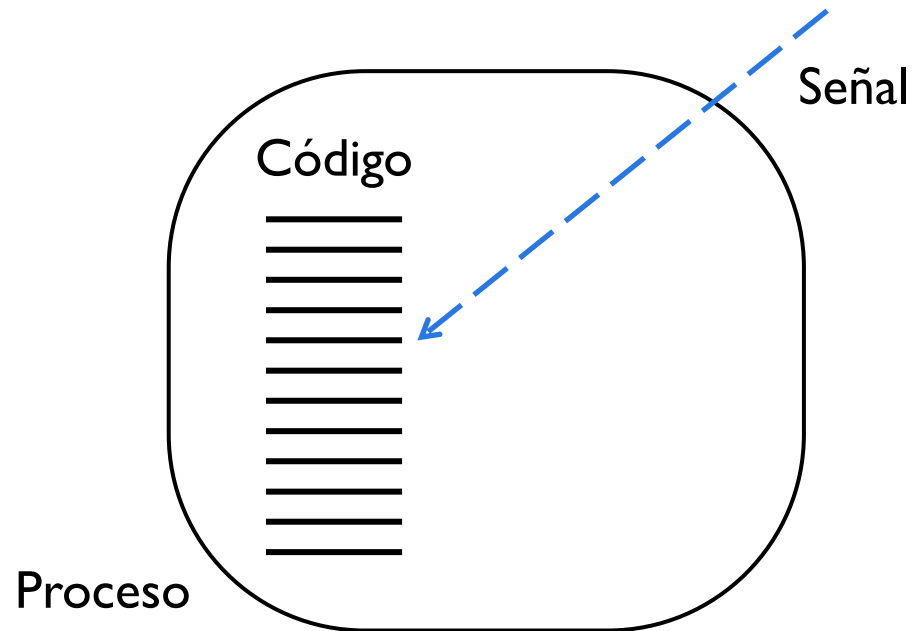
Señales: interrupciones al proceso

- ▶ Mecanismo para comunicar a un proceso la ocurrencia de un evento de forma asíncrona y permitir reaccionar a dicho evento.
 - ▶ Las señales son interrupciones al proceso.
 - ▶ Se ejecuta una función de tratamiento asociada de inmediato:



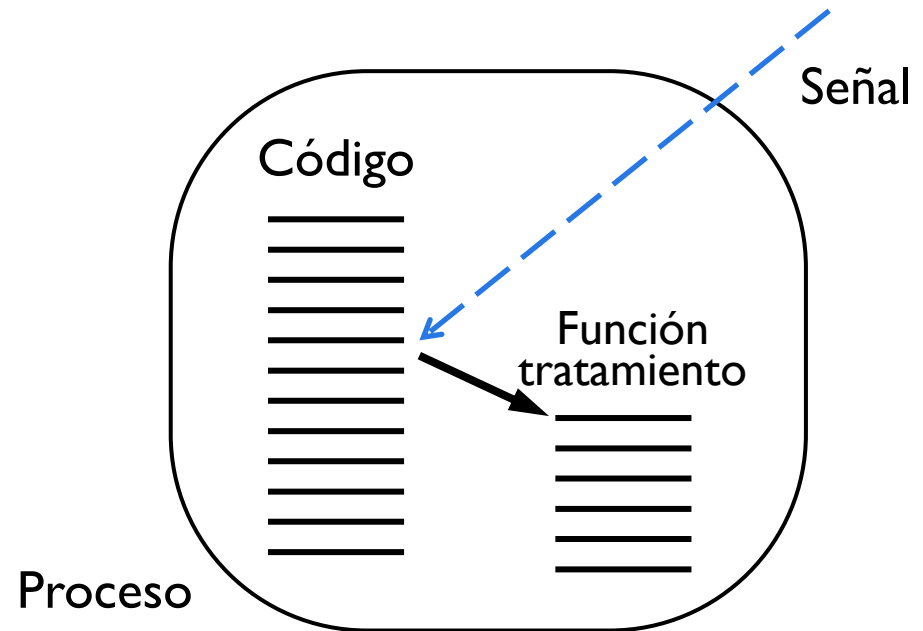
Señales: interrupciones al proceso

- ▶ Mecanismo para comunicar a un proceso la ocurrencia de un evento de forma asíncrona y permitir reaccionar a dicho evento.
 - ▶ Las señales son interrupciones al proceso.
 - ▶ Se ejecuta una función de tratamiento asociada de inmediato:



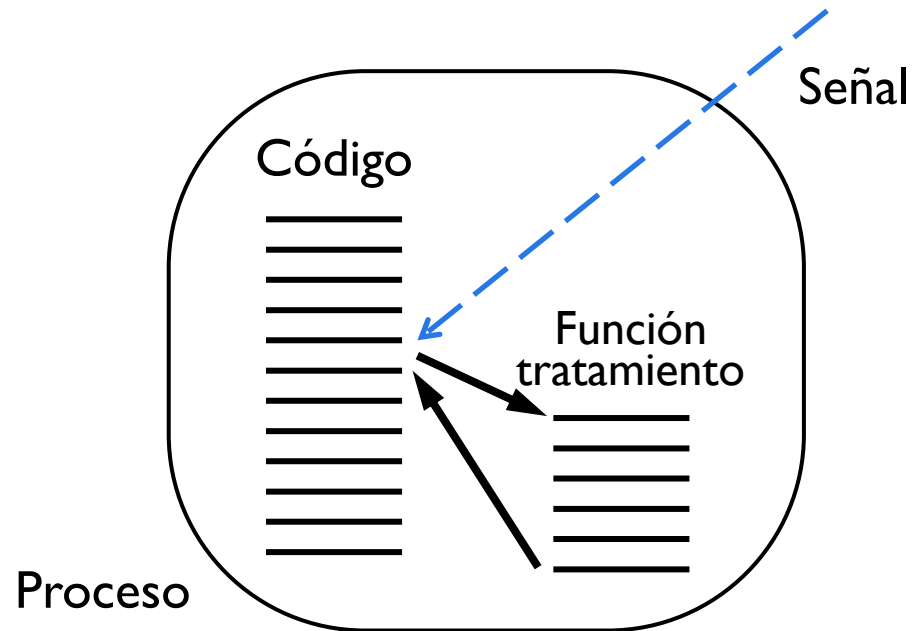
Señales: interrupciones al proceso

- ▶ Mecanismo para comunicar a un proceso la ocurrencia de un evento de forma asíncrona y permitir reaccionar a dicho evento.
 - ▶ Las señales son interrupciones al proceso.
 - ▶ Se ejecuta una función de tratamiento asociada de inmediato:



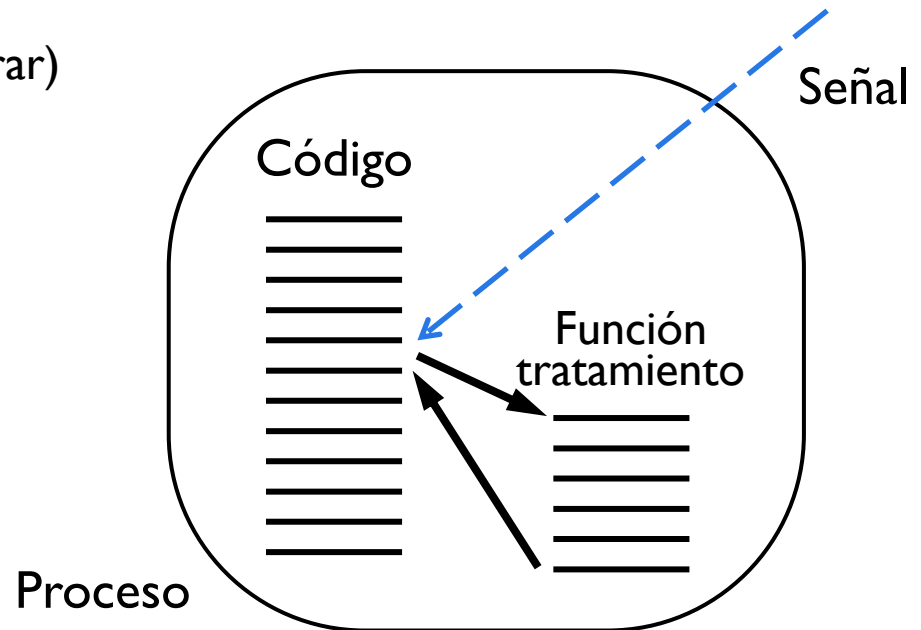
Señales: interrupciones al proceso

- ▶ Mecanismo para comunicar a un proceso la ocurrencia de un evento de forma asíncrona y permitir reaccionar a dicho evento.
 - ▶ Las señales son interrupciones al proceso.
 - ▶ Se ejecuta una función de tratamiento asociada de inmediato:



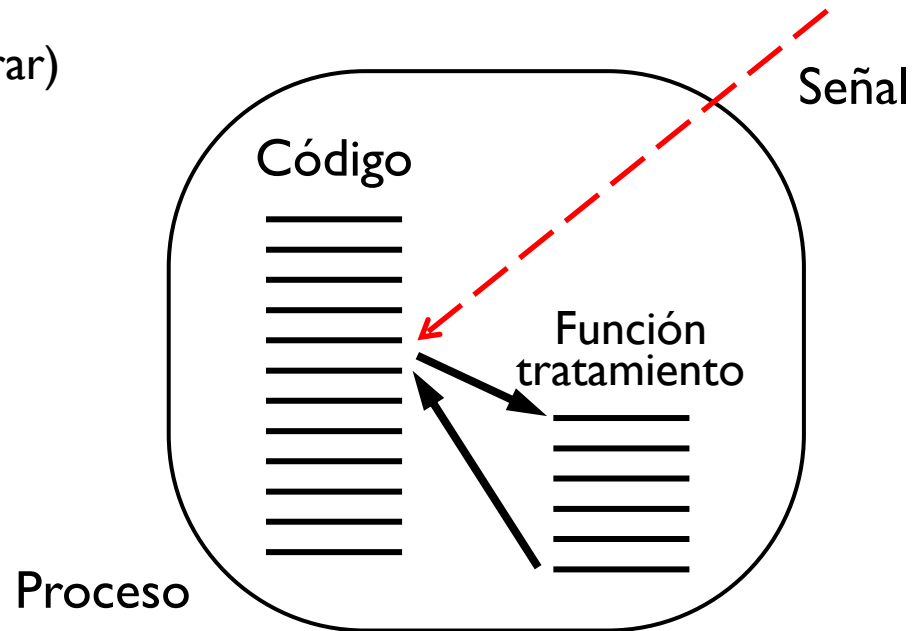
Señales: interrupciones al proceso

- ▶ Mecanismo para comunicar a un proceso la ocurrencia de un evento de forma asíncrona y permitir reaccionar a dicho evento.
- ▶ Las señales son interrupciones al proceso.
- ▶ Se ejecuta una función de tratamiento asociada de inmediato:
 - ▶ Ignorar la señal (ser “inmune)
 - ▶ Tratar por defecto (matar/ignorar)
 - ▶ Invocar a una rutina propia.



Señales: interrupciones al proceso

- ▶ Mecanismo para comunicar a un proceso la ocurrencia de un evento de forma asíncrona y permitir reaccionar a dicho evento.
 - ▶ Las señales son interrupciones al proceso.
 - ▶ Se ejecuta una función de tratamiento asociada de inmediato:
 - ▶ Ignorar la señal (ser “inmune”)
 - ▶ Tratar por defecto (matar/ignorar)
 - ▶ Invocar a una rutina propia.
- ▶ **Envío o generación desde:**
 - ▶ Sistema operativo
 - ▶ Proceso



Programación orientada a eventos

aspectos generales

```
int main ( ... )  
{  
    ...  
    On (event1, handler1) ;  
    ...  
}
```

1) Asociar el manejador
(handler1) al evento

Programación orientada a eventos

aspectos generales

```
void handler1 ( ... )  
{  
}
```

2) Codificar la función
manejador que tratará el evento

```
int main ( ... )  
{  
    ...  
    On (event1, handler1) ;  
    ...  
}
```

1) Asociar el manejador
(handler1) al evento

Programación orientada a eventos

aspectos generales

```
int global1;  
...
```

3) Para comunicar funciones,
se usa variables globales

```
void handler1 ( ... )  
{  
}
```

2) Codificar la función
manejador que tratará el evento

```
int main ( ... )  
{  
    ...  
    On (event1, handler1) ;  
    ...  
}
```

1) Asociar el manejador
(handler1) al evento

Ejemplo: contar veces se pulsa Ctrl-C

API antiguo

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

```
int contador = 0 ;
int salir = 0 ; // false
```

```
void sig_handler ( int signal_id )
{
    if (SIGINT == signal_id) {
        printf("contador = %d\n", contador);
        contador++ ;
    }
    if (SIGQUIT == signal_id) {
        salir = 1; // true
    }
}
```

```
int main ( int argc, char *argv[] )
{
    signal(SIGINT, sig_handler); // CTRL+c
    signal(SIGQUIT, sig_handler); // CTRL+\
    while(!salir) {}
    return 0 ;
}
```

3) Para comunicar funciones,
se usa variables globales

2) Codificar la función
manejador que tratará el evento

1) Asociar el manejador
(handler1) al evento

signal.h

SIGILL_____instrucción ilegal
SIGALRM_____vence el temporizador
SIGKILL_____mata al proceso
SIGSEGV_____violación segmento memoria
SIGUSR1 y SIGUSR2__reservadas para el uso del programador

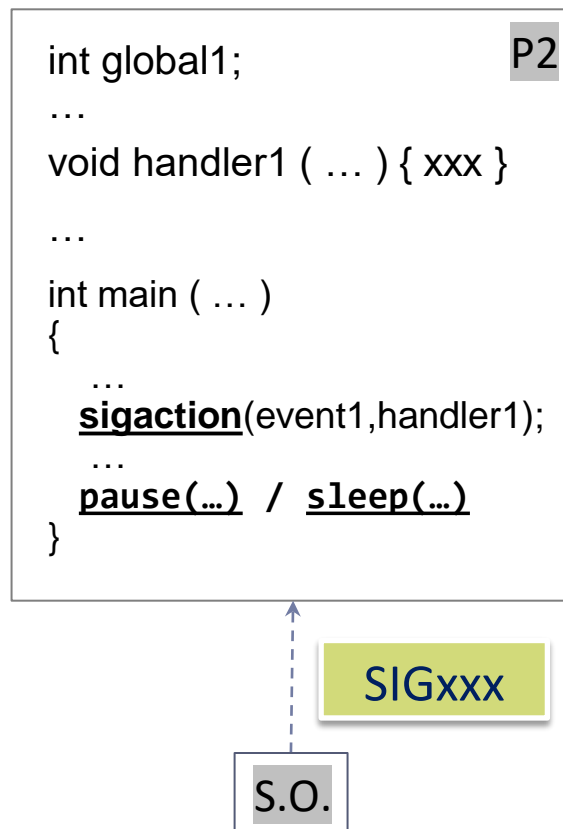
```
alex@patata:$ kill -l
```

| | | | | |
|--------------------|--------------------|--------------------|--------------------|--------------------|
| 1) SIGHUP | 2) SIGINT | 3) SIGQUIT | 4) SIGILL | 5) SIGTRAP |
| 6) SIGABRT | 7) SIGBUS | 8) SIGFPE | 9) SIGKILL | 10) SIGUSR1 |
| 11) SIGSEGV | 12) SIGUSR2 | 13) SIGPIPE | 14) SIGALRM | 15) SIGTERM |
| 16) SIGSTKFLT | 17) SIGCHLD | 18) SIGCONT | 19) SIGSTOP | 20) SIGTSTP |
| 21) SIGTTIN | 22) SIGTTOU | 23) SIGURG | 24) SIGXCPU | 25) SIGXFSZ |
| 26) SIGVTALRM | 27) SIGPROF | 28) SIGWINCH | 29) SIGIO | 30) SIGPWR |
| 31) SIGSYS | 34) SIGRTMIN | 35) SIGRTMIN+1 | 36) SIGRTMIN+2 | 37) SIGRTMIN+3 |
| 38) SIGRTMIN+4 | 39) SIGRTMIN+5 | 40) SIGRTMIN+6 | 41) SIGRTMIN+7 | 42) SIGRTMIN+8 |
| 43) SIGRTMIN+9 | 44) SIGRTMIN+10 | 45) SIGRTMIN+11 | 46) SIGRTMIN+12 | 47) SIGRTMIN+13 |
| 48) SIGRTMIN+14 | 49) SIGRTMIN+15 | 50) SIGRTMAX-14 | 51) SIGRTMAX-13 | 52) SIGRTMAX-12 |
| 53) SIGRTMAX-11 | 54) SIGRTMAX-10 | 55) SIGRTMAX-9 | 56) SIGRTMAX-8 | 57) SIGRTMAX-7 |
| 58) SIGRTMAX-6 | 59) SIGRTMAX-5 | 60) SIGRTMAX-4 | 61) SIGRTMAX-3 | 62) SIGRTMAX-2 |
| 63) SIGRTMAX-1 | 64) SIGRTMAX | | | |

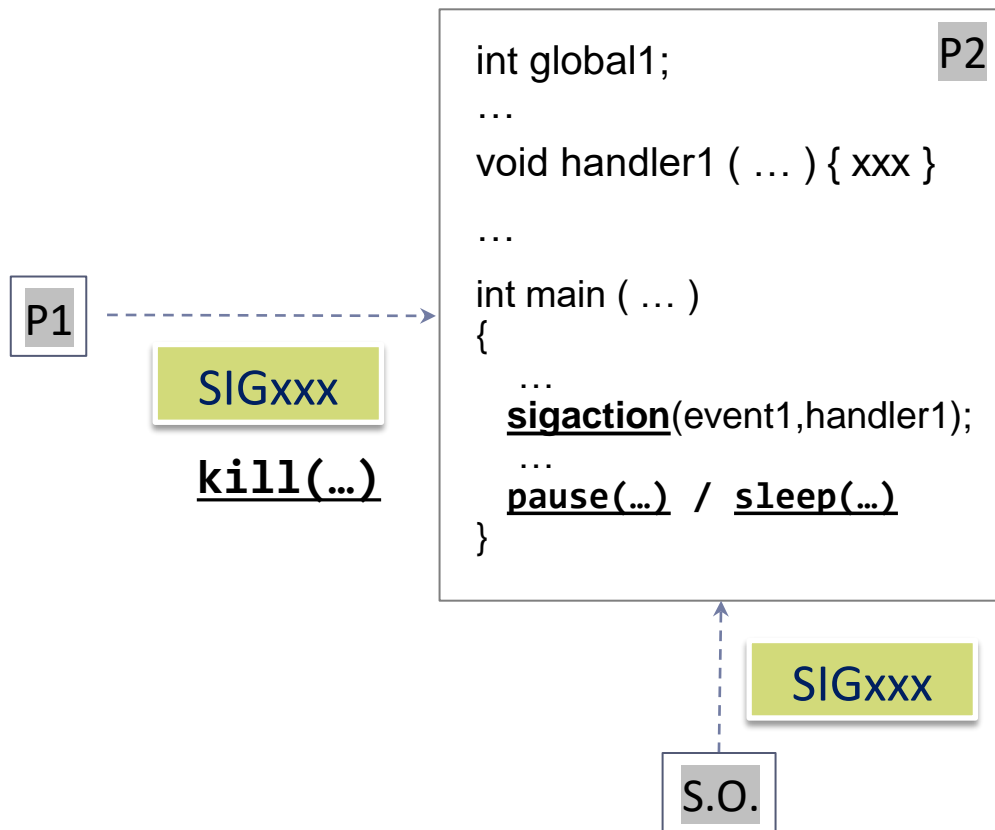
Señales: ejemplos

- ▶ Un proceso **recibe desde sistema operativo** la señal
 - ▶ SIGCHLD cuando termina un proceso hijo.
 - ▶ SIGILL cuando intenta ejecutar una instrucción máquina ilegal.
 - ▶ Un proceso cuando está ejecutando desde un terminal como tarea en primer plano y se pulsa las teclas:
 - Control y c simultáneamente recibe una señal SIGINT.
 - Control y z simultáneamente recibe una señal SIGSTOP.
- ▶ Un proceso **recibe desde otro proceso** la señal SIGUSR1 cuando el otro proceso ejecuta `kill(<pid>, SIGUSR1)` ;

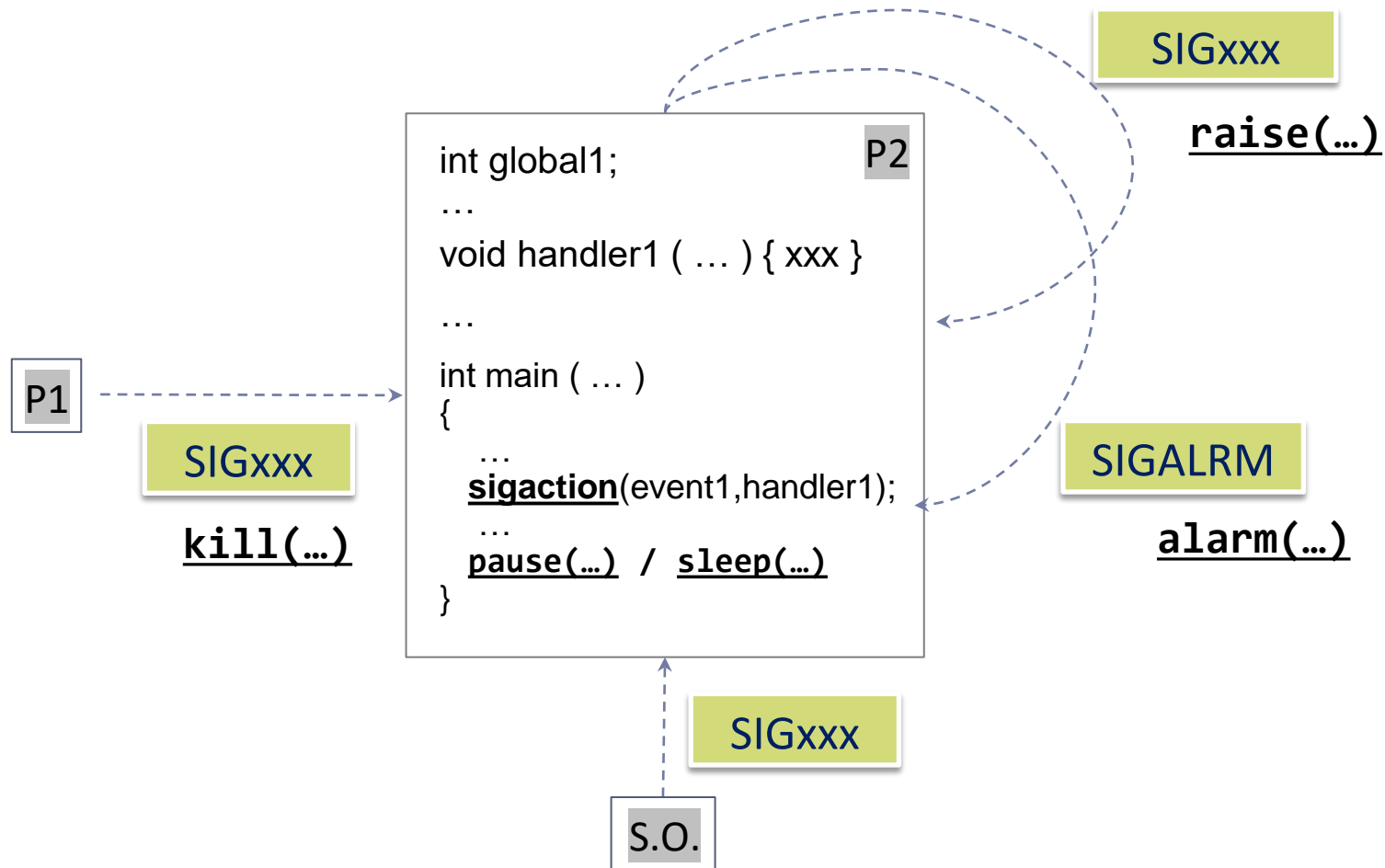
Señales: sistema operativo a proceso



Señales: proceso a proceso



Señales: proceso a sí mismo



Ejemplo: contar veces se pulsa Ctrl-C

API nuevo

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

int contador = 0 ;
int salir = 0 ; // false

void sig_handler ( int signal_id )
{
    if (SIGINT == signal_id) {
        printf("contador = %d\n", contador);
        contador++ ;
    }
    if (SIGQUIT == signal_id) {
        salir = 1; // true
    }
}

int main ( int argc, char *argv[] )
{
    struct sigaction act ;

    act.sa_handler = sigint_handler ;
    act.sa_flags = 0 ; // por defecto
    sigaction(SIGINT, &act, NULL) ; // CTRL+c
    sigaction(SIGQUIT, &act, NULL) ; // CTRL+\
    while(!salir) {}
    return 0 ;
}
```

Servicios POSIX para la gestión de señales

kill

| | |
|-------------|--|
| Servicio | <pre>int kill (pid_t pid, int sig) ;</pre> |
| Argumentos | <ul style="list-style-type: none">▣ pid: identificador de el/los proceso/s al/os que mandar la señal.▣ sig: identificador de la señal a mandar. |
| Devuelve | <ul style="list-style-type: none">▣ Cero si éxito (al menos una señal fue enviada).▣ -1 en caso de error. |
| Descripción | <ul style="list-style-type: none">▣ Envía al proceso "pid" la señal "sig".▣ Casos especiales:<ul style="list-style-type: none">▣ pid > 0 -> proceso con identificador <pid>▣ pid = 0 -> a todos los procesos con igual gid que el llamante a kill().▣ pid = -1 -> a todos los proceso que el llamante a kill() puede enviar.▣ pid < -1 -> a todos los procesos del grupo de proceso con ID <pid> |

Servicios POSIX para la gestión de señales

raise

| | |
|-------------|---|
| Servicio | <code>int raise (int sig) ;</code> |
| Argumentos | <ul style="list-style-type: none">▣ sig: identificador de la señal a mandar. |
| Devuelve | <ul style="list-style-type: none">▣ Cero si éxito (al menos una señal fue enviada).▣ -1 en caso de error. |
| Descripción | <ul style="list-style-type: none">▣ Envía al propio proceso la señal "sig".▣ En uniproseso equivale a <code>kill(getpid(), sig);</code> En multihilo a <code>pthread_kill(pthread_self(), sig);</code> |

Servicios POSIX para la gestión de señales

pause

| | |
|-------------|---|
| Servicio | <code>int pause (void) ;</code> |
| Argumentos | <ul style="list-style-type: none">□ Ninguno. |
| Devuelve | <ul style="list-style-type: none">□ Tras llegar la señal y ejecutarse su manejador, <code>pause()</code> devuelve <code>-1</code>. |
| Descripción | <ul style="list-style-type: none">□ Bloquea al proceso hasta la recepción de una señal.□ Detalles a recordar:<ul style="list-style-type: none">□ No se puede especificar un plazo para desbloqueo.□ No permite indicar el tipo de señal que se espera.□ No desbloquea el proceso ante señales ignoradas. |

Servicios POSIX para la gestión de señales

sigaction

| | |
|-------------|--|
| Servicio | <pre>int sigaction (int sig, struct sigaction *act, struct sigaction *oact) ;</pre> |
| Argumentos | <ul style="list-style-type: none">▣ sig: identifica la señal.▣ act: puntero a estructura que describe el tratamiento a usar.▣ oact: (si != NULL) apuntará al antiguo tratamiento usado antes. |
| Devuelve | <ul style="list-style-type: none">▣ Cero si éxito.▣ -1 en caso de error. |
| Descripción | <ul style="list-style-type: none">▣ Permite especificar la acción a realizar como tratamiento de la señal “sig”.▣ La configuración anterior se guarda en “oact” si no es NULL.▣ La estructura sigaction:<ul style="list-style-type: none">▣ sa_handler: SIG_DFL (por defecto, en general muere pero algún caso ignora), SIG_IGN (ignorar) o puntero a función a usar.▣ sa_sigaction: alternativa a sa_handler (no usar a la vez o ninguna).▣ sa_mask: máscara de señales a bloquear durante el manejador.▣ sa_flags: cero por defecto (conjunto de <i>flags</i>). |

Servicios POSIX para la gestión de señales

sigaction

| | |
|-------------|--|
| Servicio | <pre>int sigaction (int sig, struct sigaction *act, struct sigaction *oact) ;</pre> |
| Argumentos | <ul style="list-style-type: none">❑ sig: identifica la señal.❑ act: puntero a estructura que describe el tratamiento a usar.❑ oact: (si != NULL) apuntará al antiguo tratamiento usado antes. |
| Devuelve | <ul style="list-style-type: none">❑ Cero si éxito.❑ -1 en caso de error. |
| Descripción | <ul style="list-style-type: none">❑ Permite especificar la acción a realizar como tratamiento de la señal “sig”.❑ La configuración anterior se guarda en “oact” si no es NULL.❑ La estructura sigaction:<ul style="list-style-type: none">❑ sa_handler: SIG_DFL (por defecto, en general muere pero algún caso ignora), SIG_IGN (ignorar) o puntero a función a usar.❑ sa_sigaction: alternativa a sa_handler (no usar a la vez o ninguna).❑ sa_mask: máscara de señales a bloquear durante el manejador.❑ sa_flags: cero por defecto (conjunto de <i>flags</i>). |

API para conjuntos de señales

- ▶ `int sigemptyset (sigset_t * set) ;`
 - ▶ **Crea** un conjunto **vacío** de señales.
- ▶ `int sigfillset (sigset_t * set);`
 - ▶ **Crea** un conjunto **lleno** con todas la señales posibles.
- ▶ `int sigaddset (sigset_t * set, int signo);`
 - ▶ **Añade** una **señal** a un conjunto de señales.
- ▶ `int sigdelset (sigset_t * set, int signo);`
 - ▶ **Borra** una **señal** de un conjunto de señales.
- ▶ `int sigismember (sigset_t * set, int signo);`
 - ▶ **Comprueba si** una **señal pertenece a un conjunto**.

Servicios POSIX

sleep

| | |
|-------------|---|
| Servicio | <code>int sleep (unsigned int sec) ;</code> |
| Argumentos | <ul style="list-style-type: none">▣ sec: segundos a dormir el proceso (suspende su ejecución). |
| Devuelve | <ul style="list-style-type: none">▣ Cero si ha pasado todo el tiempo o el número de segundos que resta por dormir si el proceso ha sido interrumpido por una señal. |
| Descripción | <ul style="list-style-type: none">▣ Suspende un proceso hasta que vence un plazo o se recibe una señal |

Contenidos

1. Señales y **excepciones**.
2. Temporizadores.
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (pipes).
 - ▶ Paso de mensajes local.

Eventos del S.O.: Excepciones

- Durante el arranque.
- **Tras el arranque, se ejecuta en respuesta a eventos:**

- **Llamada al sistema.**

- { Origen: “procesos”, Función: “Petición de servicios” }
 - Gestión de procesos
 - Gestión de memoria
 - Gestión de ficheros
 - Gestión de dispositivos
 - Comunicación
 - Mantenimiento

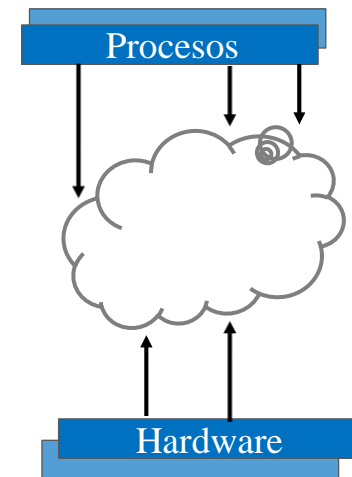
- **Excepción.**

- { Origen: “**procesos**”, Función: “**Tratar excepciones**” }

- **Interrupción hardware.**

- { Origen: “hardware”, Función: “Petición de atención del hw.” }

- En procesos de núcleo (firewall, etc.)



Excepciones

- El hardware detecta condiciones especiales:
 - ▣ División por cero, fallo de página, escritura a página de solo lectura, desbordamiento de pila, etc.
- Transfiere control al S.O. para su tratamiento.
 - ▣ Salva contexto del proceso.
 - ▣ Paso a modo protegido
 - ▣ Ejecución de manejador en el S.O.
 - ▣ La mayoría de las excepciones provoca que el S.O. envíe una señal al proceso indicando la excepción.
 - ▣ Muchos lenguajes de programación (Java, C++, Python, ...) usan un mecanismo de excepciones para manejar errores en tiempo de ejecución.

Ejemplo: excepciones en Java

```
public class JavaExceptionExample
{
    public static void main ( String args[] )
    {
        try
        {
            // array index
            int a[] = new int[2] ;
            a[5] = 20 ;

            // divide by cero
            int data=100 / 0 ;

            // ...
        }
        catch ( ArithmeticException e )
        {
            System.out.println(e);
        }

        System.out.println("After exception\n");
    }
}
```

- En Java hay una construcción para trabajar con excepciones:
Try { <código> } catch (<exception>) { <código> }

- Evitan código de comprobación en programas:
 - Mejora la claridad del código
 - Mejor rendimiento.

- En Java hay una clase para representar una excepción, con una jerarquía de subclases.

- Ejemplos: `ArrayIndexOutOfBoundsException`, `NullPointerException`, `FileNotFoundException`, `ArithmeticException`, `IllegalArgumentException`

Contenidos

1. Señales y excepciones.
2. **Temporizadores.**
3. Entorno de un proceso.
4. Comunicación de procesos con tuberías (pipes).
 - ▶ Paso de mensajes local.

Temporizadores

UNIX

- El S.O. mantiene un temporizador por proceso.
 - ▣ Se mantiene en el BCP del proceso un contador del tiempo que falta para que venza el temporizador.
- El S.O. envía una señal SIGALRM al proceso cuando vence el temporizador en curso.
 - ▣ Si un temporizador en el BCP llega a cero se ejecuta la función de tratamiento.
- El S.O. tiene un API para trabajar con temporizadores.
 - ▣ *alarm(segundos)* actualiza el contador en el BCP.

Servicios POSIX para temporización

alarm

| | |
|-------------|---|
| Servicio | <code>int alarm (unsigned int sec) ;</code> |
| Argumentos | <ul style="list-style-type: none">▣ sec: número de segundos tras los que se enviará SIGALRM. |
| Devuelve | <ul style="list-style-type: none">▣ Si no había un temporizador activado entonces devuelve cero.▣ En caso contrario devuelve el número de segundos que quedaban para vencer el temporizador anterior. |
| Descripción | <ul style="list-style-type: none">▣ Establece un nuevo temporizador:<ul style="list-style-type: none">▣ Si había un temporizador en marcha entonces se elimina y se pone uno nuevo en su lugar.▣ Si el parámetro es cero se desactiva el temporizador. |

Ejemplo 1: imprimir mensaje cada 5 seg.

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

void tratar_sigalarm ( int signal_id ) {
    printf("¡ALARMA!\n");
}

int main ( int argc, char *argv[] )
{
    struct sigaction act ;

    act.sa_handler = tratar_sigalarm ;
    act.sa_flags = 0 ; /* por defecto */
    sigaction(SIGALRM, &act, NULL) ;

    while(1) {
        alarm(5) ;
        pause() ;
    }

    return 0 ;
}
```

Ejemplo 2: finalización temporizada (1/2)

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

pid_t pid;
void tratar_alarma(void) {
    kill(pid, SIGKILL);
}

int main ( int argc, char **argv )
{
    int status;
    char **argumentos;
    struct sigaction act;
    argumentos = &argv[1];
    pid = fork();
    switch(pid) {
        case -1: /* error */
            perror ("fork");
            exit (-1);
        case 0: /* hijo */
            execvp(argumentos[0], argumentos);
            perror("exec");
            exit(-1)
        default: /* padre */
            act.sa_handler = tratar_alarma;
            act.sa_flags = 0;
            sigaction(SIGALRM, &act, NULL);
            alarm(5);
            wait(&status);
    }
    return 0;
}
```

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

```
pid_t pid;
```

```
void tratar_sigal(void)
{
    kill(pid, SIGKILL);
}
```

```
main(int argc, char **argv)
{
    int status;
    char **argumentos;
    struct sigaction act;

    argumentos = &argv[1];
    pid = fork();
```

Ejemplo 2: finalización temporizada (2/2)

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

pid_t pid;
void tratar_alarma(void) {
    kill(pid, SIGKILL);
}

int main ( int argc, char **argv )
{
    int status;
    char **argumentos;
    struct sigaction act;
    argumentos = &argv[1];
    pid = fork();
    switch(pid) {
        case -1: /* error */
            perror ("fork");
            exit (-1);
        case 0: /* hijo */
            execvp(argumentos[0], argumentos);
            perror("exec");
            exit(-1)
        default: /* padre */
            act.sa_handler = tratar_alarma;
            act.sa_flags = 0;
            sigaction(SIGALRM, &act, NULL);
            alarm(5);
            wait(&status);
    }
    return 0;
}
```

```
switch(pid) {
    case -1: /* error fork() */
        perror ("fork");
        exit(-1);
    case 0: /* hijo */
        execvp(argumentos[0],
            argumentos);
        perror("exec");
        exit(-1);
    default: /* padre */
        act.sa_handler = tratar_sigal;
        act.sa_flags = 0;
        sigaction(SIGALRM,&act,NULL);
        alarm(5);
        wait(&status);
}

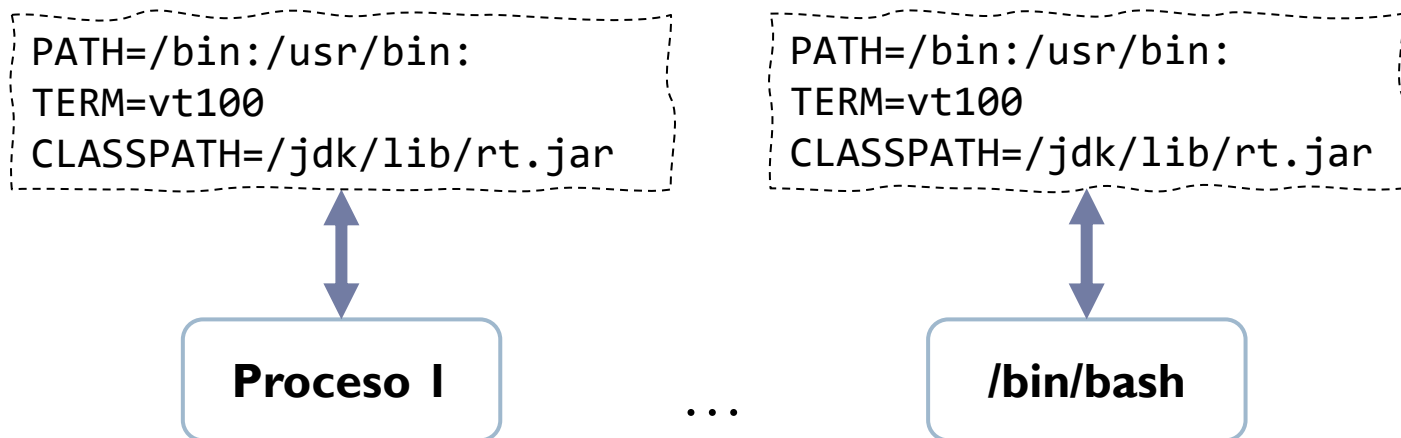
exit(0);
}
```

Contenidos

1. Señales y excepciones.
2. Temporizadores.
3. **Entorno de un proceso.**
4. Comunicación de procesos con tuberías (pipes).
 - ▶ Paso de mensajes local.

Variables de entorno

- ▶ Información accesible a un proceso en forma de tuplas clave y valor.
- ▶ Mecanismo de paso de información a procesos.
 - ▶ Puede actualizarse aspectos de configuración de los procesos
 - ▶ El propio S.O. usa las variables de entorno (ej.: PATH)



Variables de entorno

- ▶ Información accesible a un proceso en forma de tuplas clave y valor.
- ▶ Mecanismo de paso de información a procesos.
 - ▶ Puede actualizarse aspectos de configuración de los procesos
 - ▶ El propio S.O. usa las variables de entorno (ej.: PATH)
- ▶ Es posible interactuar con variables de entorno desde:
 - ▶ **Mandatos (env, set, export)**
 - ▶ En algunos S.O. + lenguajes es accesible desde main
 - ▶ API del S.O. (getenv, setenv, putenv)

Ejemplo: mandatos para entorno

```
alex@patata:$ env
SHELL=/bin/bash
PWD=/mnt/c/Users/alex
LOGNAME=alex
MOTD_SHOWN=update-motd
TERM=xterm-256color
HOME=/home/alex
USER=alex
LANG=C.UTF-8
...
```

PATH_____lista directorios donde buscar binarios
PWD_____directorio actual de trabajo
TERM_____tipo de terminal a usar en consola

```
alex@patata:$ export KEY=value
alex@patata:$ env | grep KEY
KEY=value
```

Variables de entorno

- ▶ Información accesible a un proceso en forma de tuplas clave y valor.
- ▶ Mecanismo de paso de información a procesos.
 - ▶ Puede actualizarse aspectos de configuración de los procesos
 - ▶ El propio S.O. usa las variables de entorno (ej.: PATH)
- ▶ Es posible interactuar con variables de entorno desde:
 - ▶ Mandatos (env, set, export)
 - ▶ **En algunos S.O. + lenguajes es accesible desde main**
 - ▶ API del S.O. (getenv, setenv, putenv)

Ejemplo: main con envp

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

int main ( int argc, char** argv, char** envp )
{
    int i ;

    for (i=0; envp[i] != NULL; i++)
    {
        printf("%s\n", envp[i]);
    }

    return 0;
}
```

Variables de entorno

- ▶ Información accesible a un proceso en forma de tuplas clave y valor.
- ▶ Mecanismo de paso de información a procesos.
 - ▶ Puede actualizarse aspectos de configuración de los procesos
 - ▶ El propio S.O. usa las variables de entorno (ej.: PATH)
- ▶ Es posible interactuar con variables de entorno desde:
 - ▶ Mandatos (env, set, export)
 - ▶ En algunos S.O. + lenguajes es accesible desde main
 - ▶ **API del S.O. (getenv, setenv, putenv)**

API para trabajo con entorno

- ▶ `char *getenv (const char * var) ;`
 - ▶ **Obtiene** el valor de la variable de entorno 'var'.
- ▶ `int setenv (const char * name,
const char * value,
int overwrite) ;`
 - ▶ **Modifica/Añade** una variable 'name' con valor 'value'.
- ▶ `int puntenv (const char * nombre) ;`
 - ▶ **Modifica/Añade** una variable de la forma "clave=valor"

Variables de entorno

- ▶ El entorno de un proceso hereda del padre lo siguiente:
 - ▶ Vector de argumentos con el que se invocó al programa.
 - ▶ Vector de entorno, es decir la lista de variables <nombre, valor> que el padre pasa a los hijos.
- ▶ El paso de variables de entorno entre padre e hijo:
 - ▶ Es una forma flexible de comunicar ambos y permite configurar aspectos de la ejecución del hijo (en modo usuario/a).
- ▶ El mecanismo de variables de entorno permite particularizar aspectos a nivel de cada proceso particular.
 - ▶ En lugar de tener una configuración común para todo el sistema.

Lección 3

Señales, excepciones y pipes

Sistemas Operativos
Ingeniería Informática