LLamadas al Sistema Unix

Sistemas Operativos

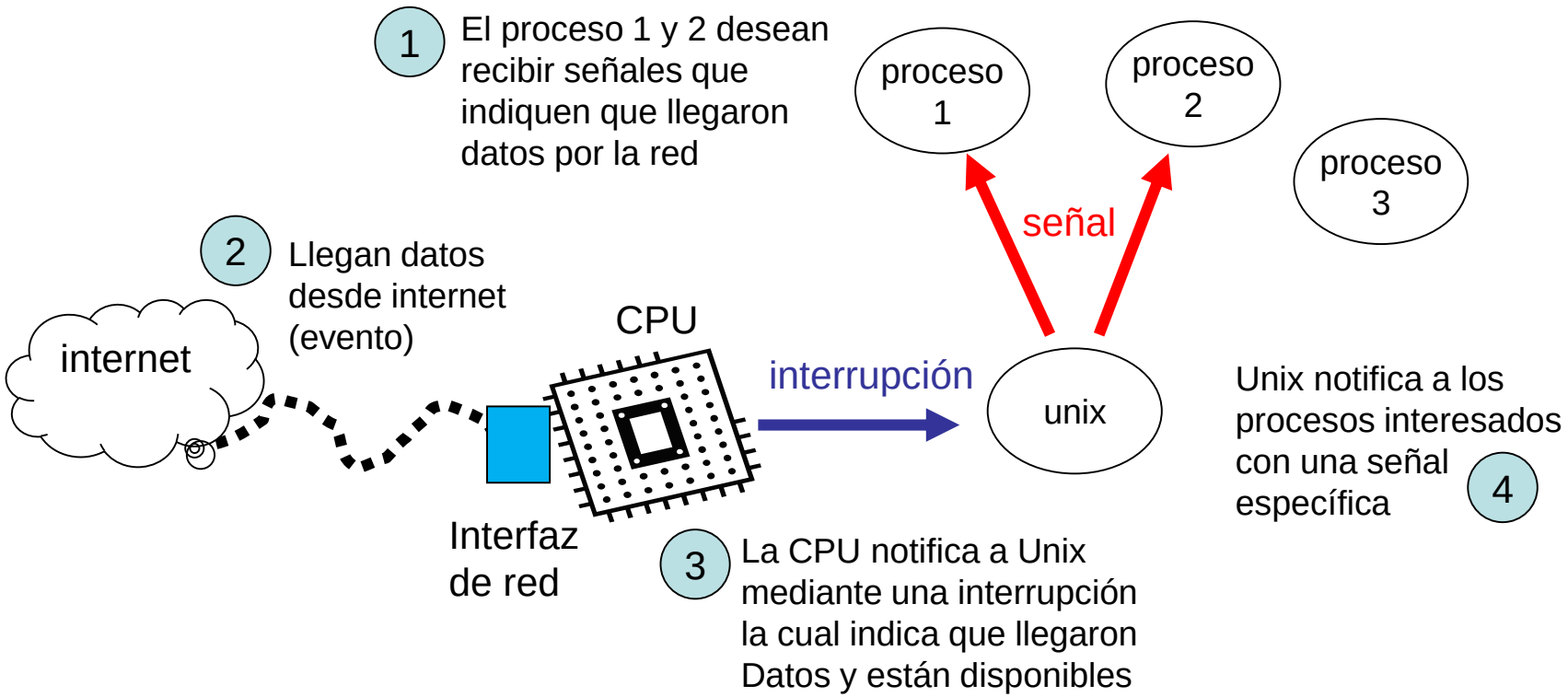
Escuela de Ingeniería Civil Informática

Señales
signal(), kill(), pause(), sleep() y alarm(),

UNIVERSIDAD DEL BÍO-BÍO

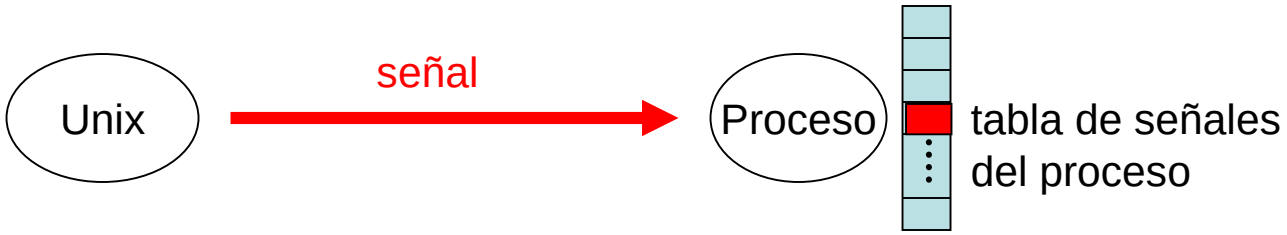
¿Qué son las señales?

- Una señal siempre es generada por el software, ya sea el kernel o un proceso de usuario.
- Una interrupción es generada por el hardware, capturada por el kernel y enviada como “señal” a todos los procesos a los que pueda afectar.



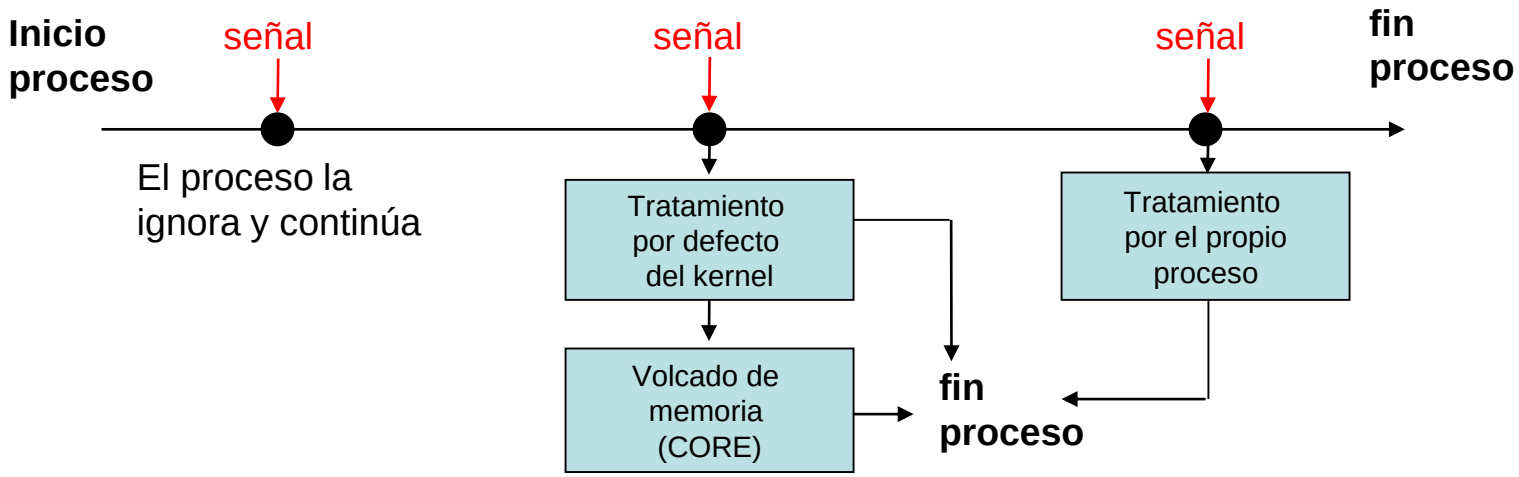
¿Qué son las señales?...

- Cuando el S.O. necesita informar a un proceso sobre un suceso o evento, lo hace por medio de una señal.
- Una señal es un tipo de comunicación muy primitivo, sin contenido explícito de información.
- Existen diferentes señales, para diferentes tipos de sucesos, por ejemplo: para indicar que ha terminado un proceso hijo, una división por cero, un error del hardware, etc.
- Cada proceso posee una tabla, la cual posee un bit por cada señal existente. Cuando se envía una señal a un proceso se enciende el bit que corresponde a esa señal para ese proceso.



¿Cómo manejar las señales?

- Un proceso puede manejar una señal de la 3 formas diferentes:
  1. Ignorar la señal, con lo cual esta no tiene efecto.
  2. Invocar la rutina manejadora de la interrupción (acción por defecto) para esa señal. Esta rutina la aporta el kernel y normalmente termina el proceso.
  3. Invocar una rutina, creada por el programador, para manejar esta señal. En este caso el programa no termina a menos que el programa lo indique.



Señales Más Relevantes

N°	Nombre	Descripción
1	SIGHUP	Colgar. Generada al desconectar el terminal
2	SIGINT	Se ha presionado ctrl + C
3	SIGQUIT	Salir. Generada por teclado
4	SIGILL	Instrucción ilegal
5	SIGTRAP	Trazado
6	SIGABRT	Abortar proceso
8	SIGFPE	Excepción aritmética, de coma flotante o división por cero
9	SIGKILL	Matar proceso. No puede capturarse, ni ignorarse
10	SIGBUS	Error en el bus (hardware)
11	SIGSEGV	Violación de segmento (muy común)
12	SIGSYS	Argumento erróneo en llamada al sistema
13	SIGPIPE	Escritura en una PIPE que otro proceso no lee
14	SIGALRM	Alarma de reloj
15	SIGTERM	Terminación del programa

Señales Más Relevantes…

N°	Nombre	Descripción
16	SIGURG	Urgencia en canal de E/S
17	SIGSTOP	Parada de proceso. No puede capturarse, ni ignorarse
18	SIGTSTP	Parada interactiva. Generada por teclado
19	SIGCONT	Continuación. Generada por teclado
20	SIGCHLD	Término de un proceso hijo
21	SIGTTIN	Un proceso en 2° plano intenta leer del terminal
22	SIGTTOU	Un proceso en 2° plano intenta escribir en el terminal
23	SIGIO	Operación de E/S completada
24	SIGXCPU	Tiempo de CPU excedido
25	SIGXFSZ	Excedido el límite de tamaño de archivo
30	SIGUSR1	Definida por el usuario (número 1)
31	SIGUSR2	Definida por el usuario (número 2)
34	SIGVTALRM	Alarma de tiempo virtual
36	SIGPRE	Excepción programada. Definida por el usuario

Enviar una Señal

- Para enviar una señal desde un proceso a otro o a un grupo de procesos se utiliza:

```
kill(pid_t pid, int sig);
```

Dónde:

pid>0 PID del proceso al cual se le envía la señal

pid=0 La señal se envía a todos los procesos del mismo grupo que el proceso que envía.

- **sig** corresponde a un valor entero que representa la señal que se quiere enviar (si es cero significa señal nula → no existe señal con ese valor). Esto sólo sirve para saber si el proceso existe.

Enviar una Señal - Ejemplo

```
#include <signal.h>

main () {
  int pid;
  if (pid=fork())==0) {
    while(1) { /*ciclo infinito*/
      printf("Soy el hijo\n");
      sleep(1);
    }
  }
  sleep(10);
  printf("Soy el padre\n");
  printf("Eliminando a mi proceso hijo\n");
  kill(pid, SIGTERM); /*También se puede usar SIGKILL*/
  exit(0);
}
```

Nota: También puede utilizarse *raise(int sig)* para enviar una señal a sí mismo. Investigar más sobre su uso.

Recibir una Señal

- Un proceso puede controlar las acciones a realizar cuando recibe una señal.

- Para realizar esto es necesario implementar una rutina de tratamiento para la señal (llamada "**manejador**") y reemplazar aquella que el sistema implementa por defecto.

- La llamada `void (*signal(int sig, void (*action)(int)))` permite realizar esto. **sig** corresponde al número de la señal y **action** indica la acción a realizar cuando se reciba la señal.

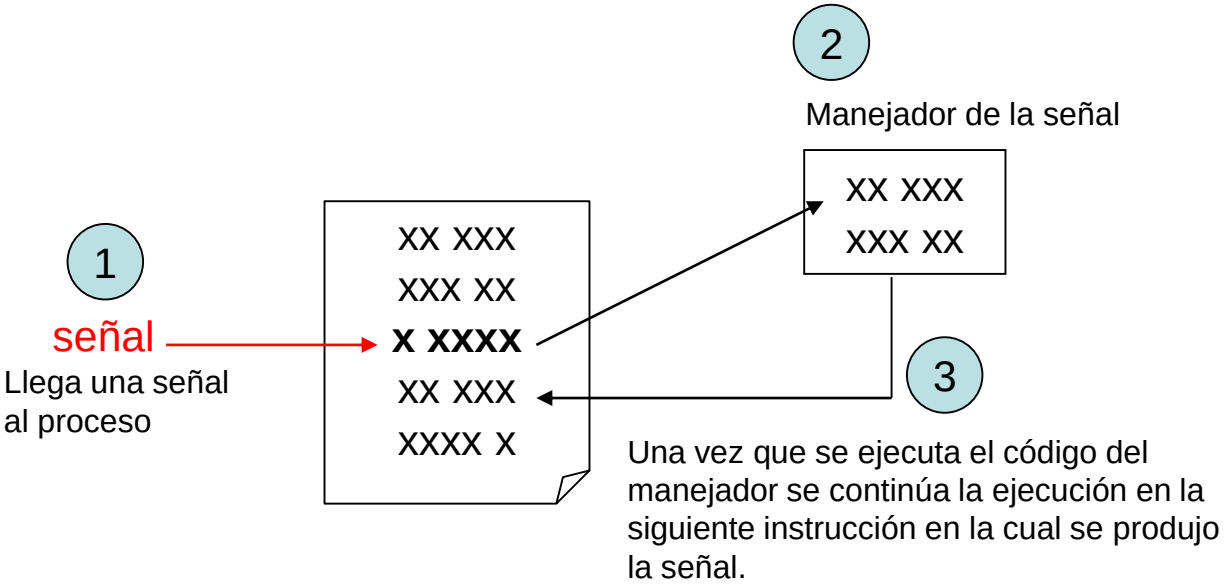
  action puede tomar uno de estos valores:

  - SIG_IGN ignorar la señal
  - SIG_DFL se debe realizar la acción por defecto (definida por el kernel)
  - dirección es la dirección de la rutina definida por el usuario (puntero a la función)

Recibir una Señal…

- La función *signal()* devuelve el valor que tenía ***action*** antes de invocarla o SIG_ERR si se produce un error.

- La llamada es **asíncrona**, esto significa que el programa continúa de inmediato y la rutina se ejecutará automáticamente cuando llegue la señal indicada.



Recibir una Señal - Ejemplo

- Ejemplo: Capturar el ctrl + C desde teclado

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

void sigint(int sig);

main() {
  if (signal(SIGINT, sigint)==SIG_ERR) { /*capturar la señal*/
    printf("Se ha producido un error\n");
    exit(-1);
  }
  while(1) {
    printf("En espera de Ctrl + C\n");
    sleep(1);
  }
}

void sigint(int sig) { /*manejador para la señal*/
  printf("señal número %d recibida \n", sig);
}
```

Recibir una Señal - Ejemplo

- El programa anterior sólo captura una vez la interrupción.

- Propuesto:

  Modifique el programa anterior para que cada vez que se presione ctrl + C el programa capture la señal.

Esperar una señal indefinidamente

- Un proceso puede esperar una señal indefinidamente, sin hacer nada, `int pause(void)` permite realizar esto.
- pause no tiene parámetros, por lo cual espera cualquier señal (no ignorada) que llegue para el proceso que la invoca.
- Una vez que llega la señal, se invoca al manejador para esa señal y luego de terminar con él se continúa con la instrucción siguiente después de pause().

NOTA: `pause()` no retorna el número de la interrupción que llegó.

Esperar una señal con pause()

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

void sigusr1(int sig);
void sigterm();

main() {
  signal(SIGTERM, sigterm);
  signal(SIGUSR1, sigusr1);
  while(1)
    pause();
}

void sigterm() { /*manejador para la señal SIGTERM*/
  printf("Terminación del proceso %d \n", getpid());
  exit(-1);
}

void sigusr1(int sig) { /*manejador para la señal SIGUSR1*/
  printf("%d\n", rand());
  signal(sig, sigusr1);
}
```

Para ejecutarlo:

```
$sigrec &       background
[1]   28302
$kill -s 15  28302
$kill -s 30  28302
```

Esperar una señal con tiempo

- Sino se desea esperar indefinidamente por una señal, se puede utilizar `unsigned int sleep(unsigned int segundos)`, la cual duerme el proceso por la cantidad indicada de segundos.

- La función `sleep()` retorna de inmediato cuando llega una señal, o cuando se cumple el tiempo si no llega una (retorna el número restante de segundos).

Programar el TIMER

- Todos los computadores contienen internamente un TIMER (reloj contador o cronómetro).
- Un proceso puede programar este TIMER para que se le avise en “t segundos” más que el tiempo se ha cumplido. La señal que obtiene el proceso es SIGALRM cuando se cumple el tiempo.
- La función *unsigned int alarm(unsigned int segundos)* nos provee exactamente la misma funcionalidad.
- Si hay una señal previamente programada, se cancela y se utiliza la nueva.

```
#include <unistd.h>
…
alarm (5);
…
```

Programar el TIMER…

- Para capturar la señal SIGALRM es necesario utilizar la función *signal()* igual como se hizo anteriormente para otras señales.

- Al utilizar *alarm()* es necesario utilizar un manejador (usar *signal()*), si esto no se hace el proceso es terminado al recibir la señal.

- Propuesto:

  Implemente un reloj mediante el TIMER, que muestre los segundos, minutos y la horas transcurridos desde que se invocó el programa.

  Investigar la función *setitimer()*.