SISTEMAS OPERATIVOS

3004610 - 1

German Sánchez Torres, I.S., M.Sc., Ph.D.

Profesor, Facultad de Ingeniería - Programa de Sistemas

Universidad del Magdalena, Santa Marta.

Phone: +57 (5) 4214079 Ext 1138 - 301-683 6593

Edificio Docente, Cub 3D401.

Email: sanchez.gt@gmail.com –gsanchez@unimagdalena.edu.co

Señales entre procesos

SISTEMAS OPERATIVOS

3004610 - 1

- Una señal es un evento generado por el sistema UNIX en respuesta a alguna condición.
- Tras la recepción de una señal, el proceso puede tomar alguna acción.
- Generación de la señal:
  - por condiciones de error (violación de segmento de memoria, error de punto flotante, instrucciones ilegales)
  - por los Shell y los manejadores de terminal
  - por un proceso a otro para notificar evento
- En todos los casos, la interfaz de programación es la misma.

- El archivo <signal.h> contiene la lista de señales posibles que se pueden enviar a los procesos. La acción ante ellas, por defecto, es finalizar el proceso.
- Todos los nombre de señales comienzan con “SIG”
- Linux soporta todas las señales definidas en POSIX.1
- Para obtener una lista de todas las señales soportadas ejecute:

```
Host@> man -S 7 signal
```

Signal Name	Description
SIGABORT	*process abort
SIGALRM	Alarm Clock
SIGFPE	*Floating point exception
SIGHUP	Hangup
SIGILL	*Illegal instruction
SIGINT	Terminal interrupt
SIGKILL	Kill (can't be caught or ignored)
SIGPIPE	Write on pipe with no reader
SIGQUIT	Terminal quit
SIGSEGV	*Invalid memory segment access
SIGTERM	Termination
SIGUSR1	User-defined signal 1
SIGUSR2	User-defined signal 2

Esta señal no puede ser capturada

Signal Name	Description
SIGCHLD	Child process has stopped or exited
SIGCONT	Continue executing, if stopped
SIGSTOP	Stop executing. (Can’t be caught or ignored)
SIGTSTP	Terminal stop signal
SIGTTIN	Background process trying to read.
SIGTTOU	Background process trying to write.

Name	Signal number	Description	SUSv3	Default
SIGABRT	6	Abort process	•	core
SIGALRM	14	Real-time timer expired	•	term
SIGBUS	7 (SAMP=10)	Memory access error	•	core
SIGCHLD	17 (SA=20, MP=18)	Child terminated or stopped	•	ignore
SIGCONT	18 (SA=19, M=25, P=26)	Continue if stopped	•	cont
SIGEMT	undef (SAMP=7)	Hardware fault		term
SIGFPE	8	Arithmetic exception	•	core
SIGHUP	1	Hangup	•	term
SIGILL	4	Illegal instruction	•	core
SIGINT	2	Terminal interrupt	•	term
SIGIO / SIGPOLL	29 (SA=23, MP=22)	I/O possible	•	term
SIGKILL	9	Sure kill	•	term
SIGPIPE	13	Broken pipe	•	term
SIGPROF	27 (M=29, P=21)	Profiling timer expired	•	term
SIGPWR	30 (SA=29, MP=19)	Power about to fail		term
SIGQUIT	3	Terminal quit	•	core
SIGSEGV	11	Invalid memory reference	•	core
SIGSTKFLT	16 (SAM=undef, P=36)	Stack fault on coprocessor		term
SIGSTOP	19 (SA=17, M=23, P=24)	Sure stop	•	stop
SIGSYS	31 (SAMP=12)	Invalid system call	•	core
SIGTERM	15	Terminate process	•	term
SIGTRAP	5	Trace/breakpoint trap	•	core
SIGTSTP	20 (SA=18, M=24, P=25)	Terminal stop	•	stop
SIGTTIN	21 (M=26, P=27)	Terminal read from BG	•	stop
SIGTTOU	22 (M=27, P=28)	Terminal write from BG	•	stop
SIGURG	23 (SA=16, M=21, P=29)	Urgent data on socket	•	ignore
SIGUSR1	10 (SA=30, MP=16)	User-defined signal 1	•	term
SIGUSR2	12 (SA=31, MP=17)	User-defined signal 2	•	term
SIGVTALRM	26 (M=28, P=20)	Virtual timer expired	•	term
SIGWINCH	28 (M=20, P=28)	Terminal window size change		ignore
SIGXCPU	24 (M=30, P=33)	CPU time limit exceeded	•	core
SIGXFSZ	25 (M=31, P=34)	File size limit exceeded	•	core

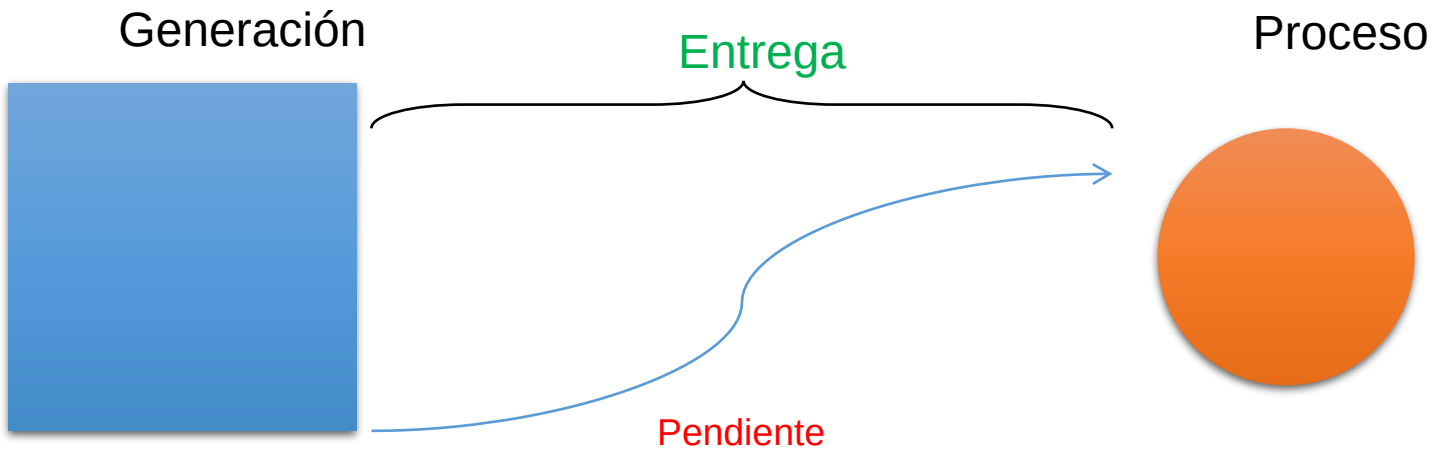
Acciones por defecto ante un señal

- The signal is *ignored*; that is, it is discarded by the kernel and has no effect on the process. (The process never even knows that it occurred.)
- The process is *terminated* (killed). This is sometimes referred to as *abnormal process termination*, as opposed to the normal process termination that occurs when a process terminates using *exit()*.
- A *core dump file* is generated, and the process is terminated. A core dump file contains an image of the virtual memory of the process, which can be loaded into a debugger in order to inspect the state of the process at the time that it terminated.
- The process is *stopped*—execution of the process is suspended.
- Execution of the process is *resumed* after previously being stopped.

Acciones por defecto ante un señal

- The *default action* should occur. This is useful to undo an earlier change of the disposition of the signal to something other than its default.
- The signal is *ignored*. This is useful for a signal whose default action would be to terminate the process.
- A *signal handler* is executed.

Entre el tiempo de *generación* de la señal y el tiempo de *entrega* se dice que la señal está *pendiente*.
Las señales pendientes son entregadas al proceso tan pronto como éste sea planificado para ejecutarse o inmediatamente si el proceso está en ejecución.

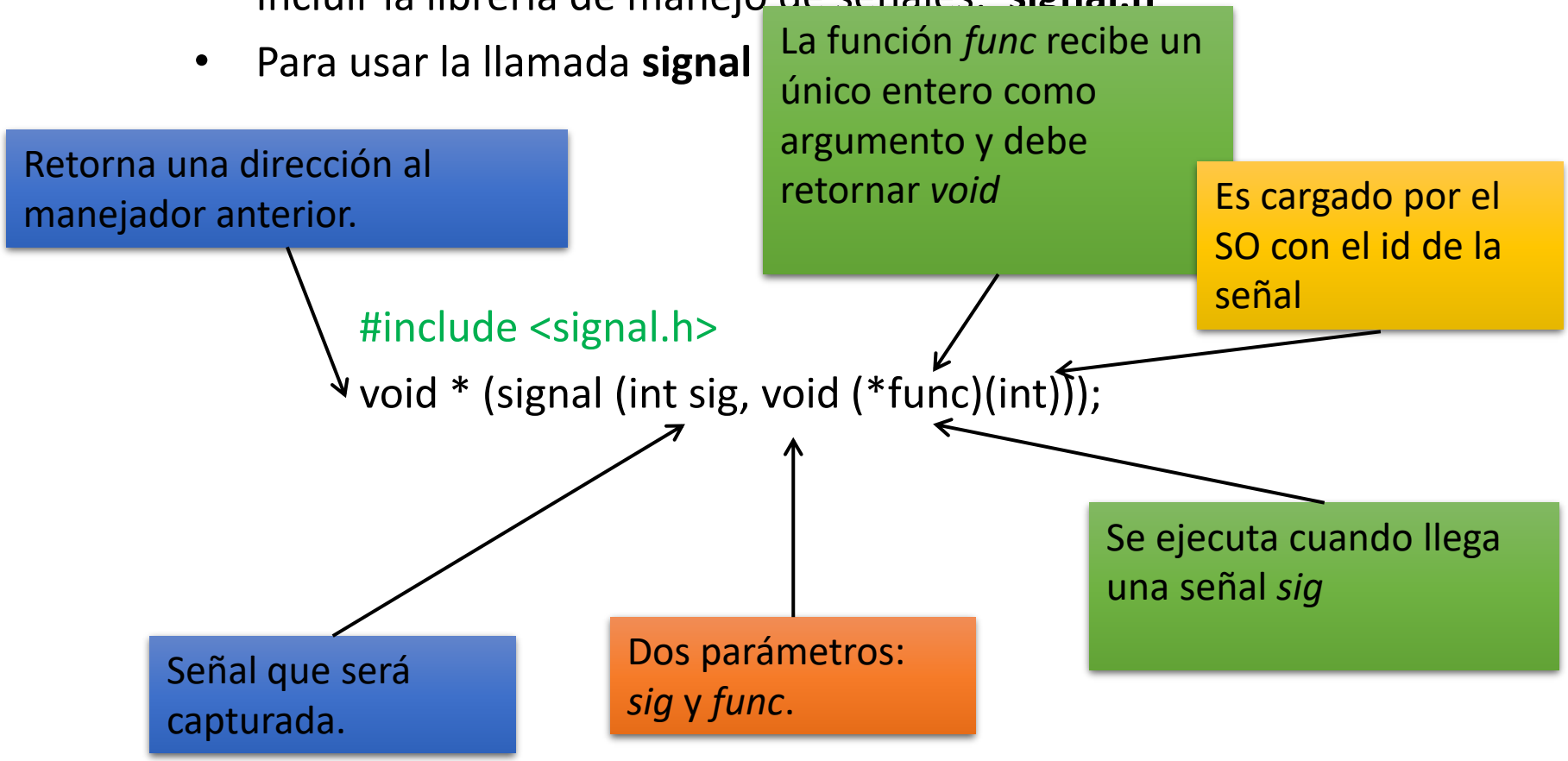


Captura de señales:

- Se puede utilizar la llamada del sistema *signal* para definir una función (que ya existe en su programa) que captará una señal específica

- También se puede utilizar la llamada al sistema *sigaction* para definir una función (que ya existe en su programa) que captará una señal específica

- Incluir la librería de manejo de señales: **signal.h**
- Para usar la llamada **signal**



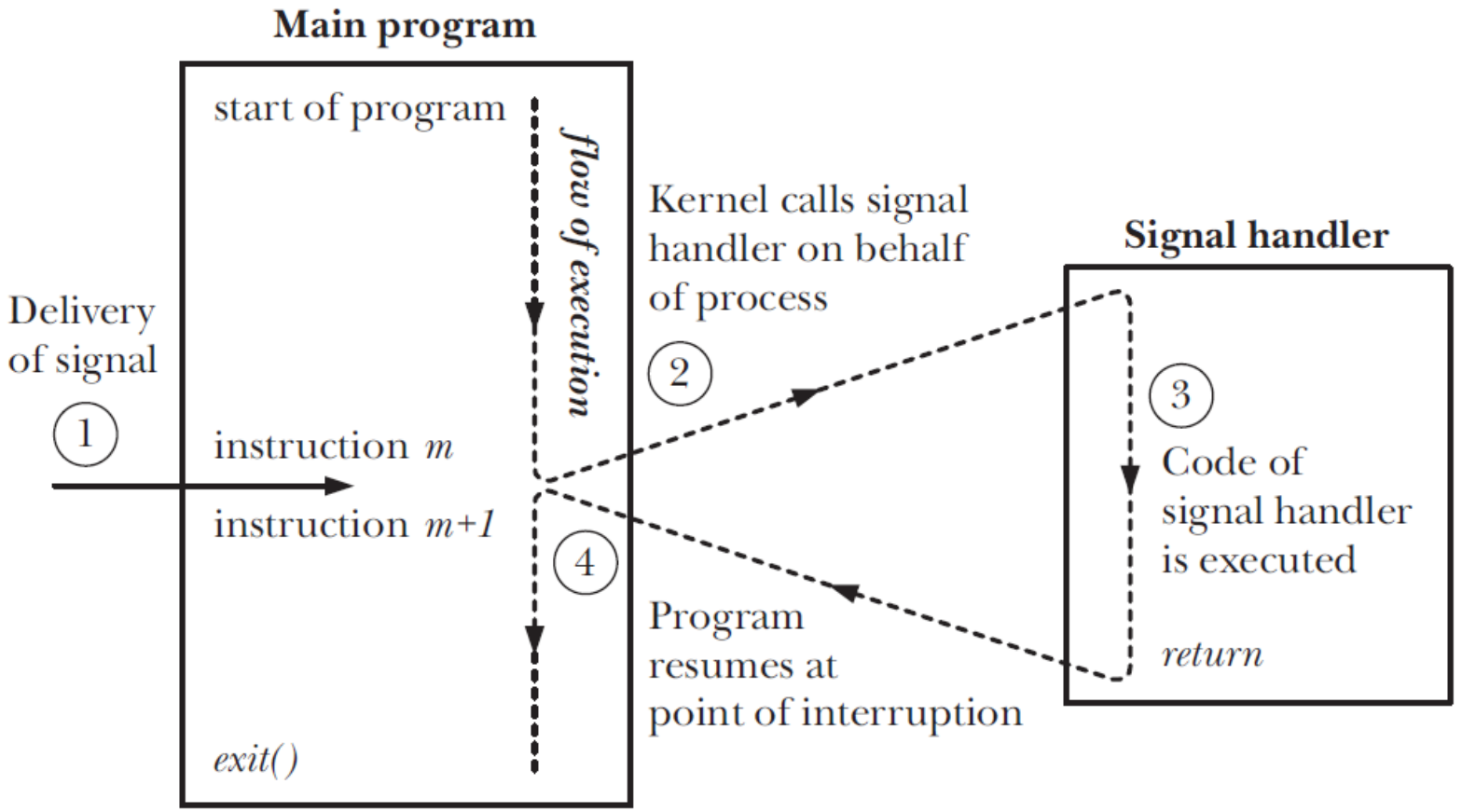
Cuando la función retorna el valor del campo *func* es establecido con el valor previo a la llamada *signal*. Esto constituye el comportamiento por defecto y puede variar de acuerdo a la distribución del SO.

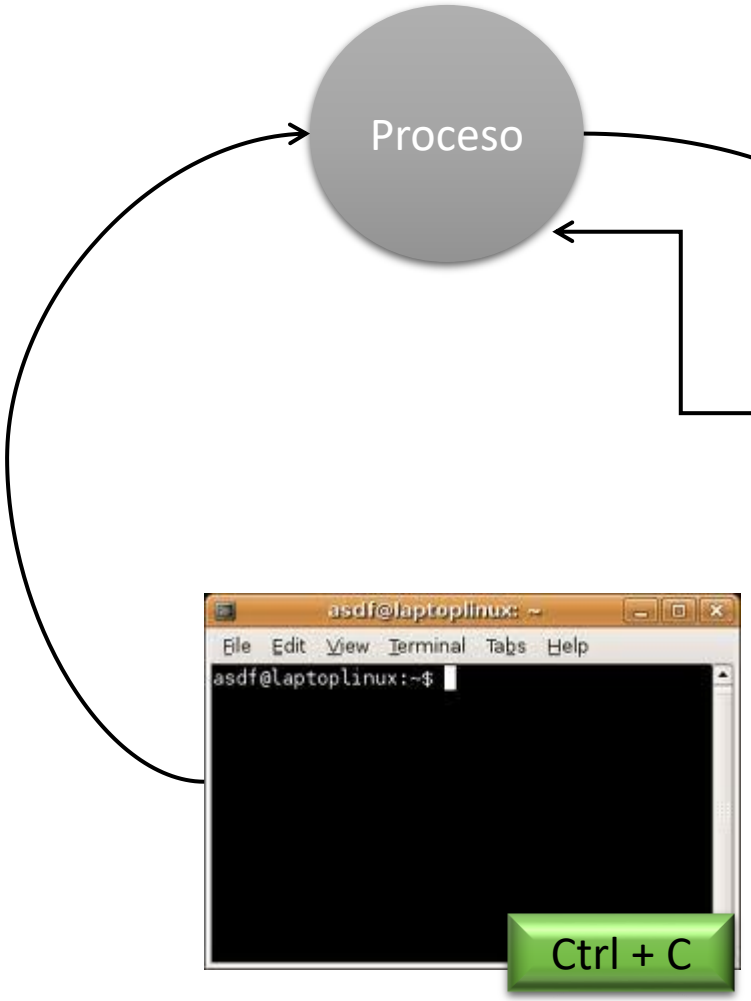
Adicionalmente, el valor de *func* puede tomar dos valores especiales:

SIG_IGN significa que la señal se ignorará.

SIG_DFL significa que se restaura el comportamiento por defecto.

Entrega y manejo de señales






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

void manejador(int sig){
    printf("Capturada señal %d\n", sig);
}
int main(){
    (void) signal(SIGINT, manejador);
    while(1) {
        printf("Hello World!\n");
        sleep(1);
    }
}
```

Envío de señales

Algunas señales son enviadas por el SO, para notificar determinados eventos. Otras pueden usarse de unos procesos a otros, mediante el servicio

int kill(pid_t pid, int señal);

Este servicio envía una señal al proceso especificados por *pid*.

Para que un proceso pueda enviar una señal a un proceso designado por *pid*, el identificador de usuario efectivo o real del proceso que envía ha de coincidir con el identificador real o efectivo del proceso que la recibe.

Temporización

El inicio de un temporizador se realiza mediante el servicio

```
unsigned int alarm (unsigned int seconds);
```

Esta función provoca que se genere la señal SIGALRM tras pasar los segundos especificados en *seconds*. Si este número es 0, se desactiva la alarma, si estuviera previamente activa.

Espera de señales

Si se desea esperar la recepción de una señal se usa el servicio pause, que bloquea el proceso hasta que se reciba.

```
int pause (void);
```

Este servicio no permite indicar qué señal se espera, por lo que la recepción de cualquier señal no bloqueada provocará del desbloqueo del proceso.

Si se desea suspender un proceso se usa:

```
int sleep(unsigned int seconds);
```

El proceso se bloquea hasta que trascurra el tiempo indicado o se reciba una señal.

Ejercicio: hacer un programa y hacer uso de **alarm** para conseguir que se ejecute la llamada a sistema ***system("date")*** cada 2 seg.
Hacer que durante la ejecución de esta función se ignore cualquier señal Ctrl+C recibida

Ejercicio: un proceso temporiza la ejecución de un proceso hijo. El proceso padre crea un proceso hijo que ejecuta un comando pasado como argumento y espera su finalización. Hace también lo necesario para que, si el hijo no finaliza en determinado tiempo, se envíe la señal KILL al hijo.

```c
/*La funcion raise( int signo) genera una señal <signo> al
proceso que llama la función*/

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

int main(){
    int sig;
    printf("Ingrese la senal a generar\n");
    scanf("%d", &sig);

    printf("Generando la senal %d\n", sig);
    raise(sig);
    printf("Señal generada\n");
    return 0;
}
```

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

void manejador(){}

int main(){
    int c;
    signal(SIGUSR1, manejador);
    if( !(c=fork()) ){
     pause();
     printf("Hijo\n");}
    else{
     printf("padre \n");
     kill(c, SIGUSR1);
     wait(NULL);}
    return  0;
}
```

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

void manejador(){}
int main(){
      int c, n,i;
      signal(SIGUSR1, manejador);
      printf("Digite n=");
      scanf("%d",&n);
      if( !(c=fork()) ){
        for(i=0; i<n; i++){
          pause();
          printf("Hijo\n");
          kill(getppid(), SIGUSR1);
        }
      }
      else{
        for(i=0; i<n; i++){
          printf("padre \n");
          kill(c, SIGUSR1);
          pause();
        }
        wait(NULL);
      }
      return  0;
}
```