Programación Avanzada (TC2025)

Tema 4. Eventos y señales

Instituto Tecnológico y de Estudios Superiores de Monterrey. Campus Santa Fe
Departamento de Tecnologías de Información y Electrónica
Dr. Vicente Cubells (vcubells@itesm.mx)

Temario

- Conceptos básicos de señales y las funciones correspondientes
- Interrupción de la ejecución de las llamadas al sistema
- Señales para el control de tareas (jobs)

Señales…

- Son **interrupciones de software que pueden enviarse a un proceso** y que proporcionan un medio para tratar eventos asíncronos
- Pueden ser generadas:
  - Por el kernel: se producen cuando se detecta un **error de software** o **de hardware** en la ejecución del proceso
    - cuando un proceso intenta acceder a una zona de memoria que no le corresponde,
    - cuando se intenta realizar una división por cero,
    - etc.

Señales

- Pueden ser generadas:
  - Por un usuario desde la línea de comando
    - Se pueden producir por la acción de pulsar ciertas teclas en un terminal
    - Mediante un comando: `kill`
    - Permite que el usuario envíe señales a un proceso o grupo de procesos
  - Por cualquier proceso que conozca el PID de otro al que quiere enviarle una señal
    - Las funciones `kill()` y `killpg()` se utilizan para la generación de señales en un proceso mediante las cuales éste envía señales a otro proceso o grupo de procesos

Proporcionan dos utilidades

- **Defensa del proceso frente a incidencias comunicadas por el kernel**
  - El kernel envía señales al proceso cuando se ha producido alguna eventualidad; esto puede suceder en cualquier momento. Si las señales no son gestionadas (ignoradas o capturadas) por el proceso al que van dirigidas, causan (casi todas) la terminación del mismo
  - Esto puede dar como resultado una pérdida irrecuperable de datos
- **Mecanismo de comunicación entre dos procesos**

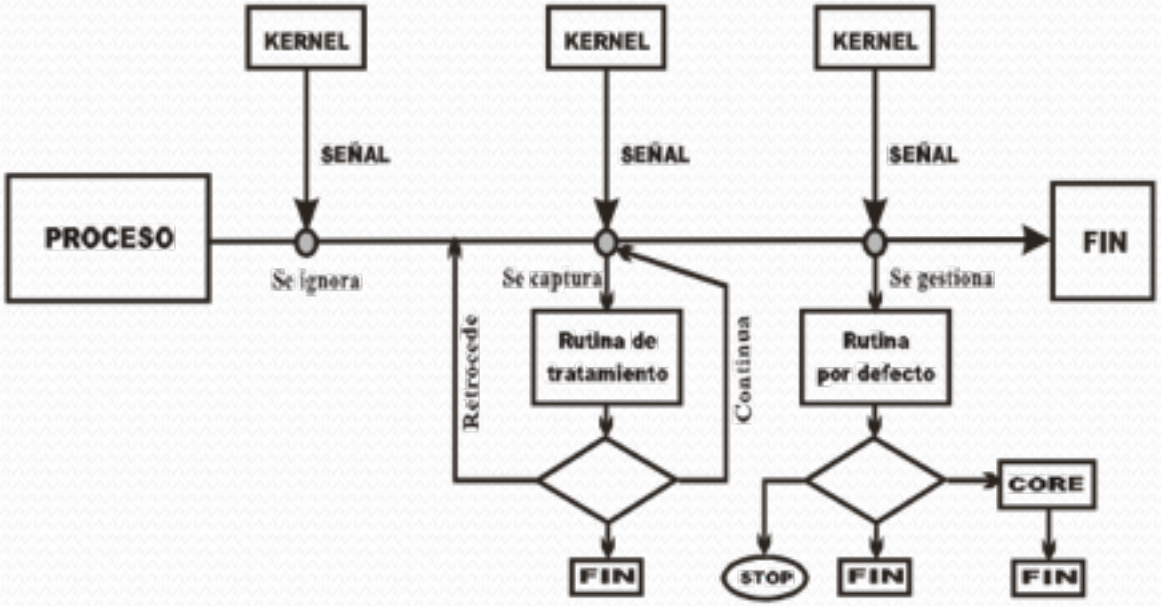
Diferencias con los pipes

- Una señal puede ser enviada en cualquier momento desde cualquier otro proceso o desde el kernel como resultado de la ocurrencia de un evento
- El proceso puede capturar la señal y realizar una acción asociada a ella, o bien ignorarla
- Las señales no tienen contenido, ni siquiera el emisor de la señal

Acciones ante una señal…

- Ignorarla
  - No todas las señales pueden ser ignoradas, esto sería catastrófico
  - SIG_IGN
- Capturarla
  - Se atiende mediante un manipulador de la señal
  - No se ejecutan en el orden secuencial de la aplicación
- Aplicar una función predeterminada
  - Exit, Core, Stop
  - SIG_DFL

Acciones ante una señal



Tipos de señales <signal.h>

Nombre simbólico	Nº	Descripción	Por defecto		
			T	C	I
SIGHUP	1	***Hangup***. Se envía a cada proceso cuyo terminal de control se desconecta o se ha quedado colgado. También se envía a cada proceso de un grupo cuando el *leader* del grupo finaliza su ejecución.	X		
SIGINT	2	***Interrupción***. Es enviada a cada proceso perteneciente al grupo de procesos que está ejecutándose en primer plano y que está asociado al terminal de control desde el que se presiona **Ctrl + C**	X		
SIGQUIT	3	***Quit***. Similar a SIGINT, siendo enviada cuando se presiona tecla de salida [Impr Pant - Pet Sis].	X	X	
SIGILL	4	***Instrucción ilegal***. Se envía cuando *el hardware* detecta un instrucción ilegal. Por ejemplo, en programas C, cuando se emplean punteros a funciones no iniciados correctamente (no apuntan a una función) y se trata de ejecutar el código de la función .	X	X	
SIGFPE	8	Se envía a un proceso cuando el hardware detecta un error aritmético, como un desbordamiento, un formato desconocido o una división por cero.	X	X	
SIGKILL	9	***Kill***. Es la única manera segura de matar un proceso, ya que la señal es siempre fatal. **NO PUEDE SER IGNORADA NI CAPTURADA.**	X		
SIGUSR1	10	***Señal de usuario 1***. Esta señal puede ser utilizada por las aplicaciones de usuario para intercomunicar procesos (no recomendado su uso para ello).	X		

Posibles acciones por defecto: **T = Terminar el proceso** **C=Genera fichero core** **I=Ignorar**

Tipos de señales <signal.h>

Nombre simbólico	Nº	Descripción	Por defecto		
			T	C	I
SIGSEGV	11	***Violación de segmento***. Es enviada cuando un proceso intenta acceder a una posición de memoria que está fuera de su espacio de direcciones.	X	X	
SIGUSR2	12	***Señal de usuario 2***. Esta señal puede ser utilizada por las aplicaciones de usuario para intercomunicar procesos (no recomendado su uso para ello).	X		
SIGPIPE	13	Es enviada cuando un proceso intenta realizar una operación de escritura en una tubería (o en un socket) en los que no hay nadie leyendo.	X		
SIGALRM	14	***Alarma de reloj***. Es enviada cuando expira un reloj asociado a un proceso - que se establece utilizando la llamada al sistema *alarm()* - .	X		
SIGTERM	15	***Terminación de software***. Es la señal utilizada para indicarle a un proceso que debe terminar su ejecución. Se genera por defecto por la función *kill()*. Esta señal no es tan tajante como SIGKILL y **puede ser ignorada**. Cuando el proceso la captura, deberá cerrar todos sus ficheros temporales y hacer un llamada a *exit()* Es enviada a todos los procesos durante la parada del sistema o *shutdown*.	X		
SIGCHLD	17	***Muerte de un hijo***. Es enviada al proceso padre cuando el hijo finaliza.			X
SIGCONT	18	La ejecución del proceso debe continuar si se había detenido.			X
SIGSTOP	19	Indica a un proceso que debe detenerse (podrá continuar si se le envía SIGCONT). **NO PUEDE SER IGNORADA NI CAPTURADA.**			
SIGTSTP	20	Se genera cuando se pulsa **Ctrl + Z** y es enviada a todos los procesos del grupo de procesos que están listos para ser ejecutados. Su acción por defecto es detener la ejecución del proceso.			

Tratamiento de señales

<signal.h>

- La función del sistema:

```
void (*signal (int signum, void (*fmanejador)
(int))) (int)
```

`signum:` número de la señal

`*fmanejador:` `SIG_IGN`, `SIG_DFL` o apuntador a

```
void fmanejador(int signum)
```

- Cada llamada a `signal()`establece el manejador para una señal. Debe existir una llamada a `signal()` por cada señal a la cual la aplicación quiera responder sin terminar

Captura señal SIGINT generada al pulsar CTRL + C

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


void gestor_ctrlc (int senial)
{
        printf("***(%d)*** Pulsó CTRL + C (señal número %d) \n", getpid(), senial);
}

int main (void)
{
        int k=10;

        if ( signal (SIGINT, gestor_ctrlc) == SIG_ERR ) {
                printf("Error en el gestor de señales.\n");
                exit(-1);
        }

        while ( k-- > 0)
        {
                printf("(%d)PID=%d. En espera de CTRL + C\n", k, getpid() );
                sleep(1);                       //ocasionar una espera
        }
}
```

Análisis de otros ejemplos

- Capturar Ctrl + C con tres posibilidades
- Capturar Ctrl + C y Ctrl + Z (dos señales)
- Capturar una señal y luego restaurar su manejador original

Resumiendo

- Las señales son interrupciones
- Permiten manejar eventos asíncronamente
- Se pueden considerar como un mecanismo de comunicación entre procesos, aunque no es recomendable este uso (no se pueden pasar valores)
- Tres maneras de manipular señales
- La función `signal()` permite especificar como responderá una aplicación ante una señal determinada