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

- Modificando manejadores de señales
- Comunicación de señales entre procesos
- Temporización de procesos
- Bloqueando señales
- Detección de señales pendientes

La función `sigaction()`

`<signal.h>`

- Permite modificar o examinar la acción asociada con una señal determinada
- Resulta de gran utilidad, puesto que permite averiguar la disposición del proceso respecto a una determinada señal sin modificarla

```
int sigaction (int signum, const struct sigaction *act,
                           struct sigaction *oldact);
```

- Cualquier señal excepto SIGKILL y SIGSTOP

struct sigaction <signal.h>

```
struct sigaction {
        void (*sa_handler)(int);
        sigset_t sa_mask;
        int sa_flags;
};
```

sa_handler : representa la dirección de una función que maneje la señal (el gestor de señales) o SIG_IGN (para ignorar la señal) o SIG_DFL (acción por defecto)

sa_mask : proporciona un máscara para las señales que deban ser bloqueadas durante la ejecución del gestor de señales

sa_flags : es la combinación de ninguno o alguno/s (si son varios se combinan mediante el operador OR) de los siguientes flags:

struct sigaction <signal.h>

flag	Descripción
SA_NOCLDSTOP	Si *signum* es SIGCHLD, no se recibe notificación cuando los procesos hijos se paren (es decir, cuando éstos reciban una de las señales SIGSTOP, SIGTSTP, SIGTTIN o SIGTTOU).
SA_ONESHOT	**Restablece** la acción correspondiente a la señal **a su valor por defecto** una vez que se ha llamado al gestor de señales.
SA_RESTART	Opuesta a SA_ONESHOT. **Restablece** la acción de la señal (que es el comportamiento de Linux por defecto).
SA_NOMASK	No previene a la señal de ser recibida dentro de su propio gestor de señales

Ver ejemplo: t4c3e1

Ver ejemplo: t4c3e2

La función pause()

<signal.h>

- Permite suspender la ejecución de un proceso hasta que éste reciba una señal
- Se dice, entonces, que el proceso queda en espera y no hace nada; como la llegada de una señal interrumpe este estado de espera, se dice que pause() espera la llegada de una señal
- No es selectiva respecto al tipo de señal que termine con su espera

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

Ver ejemplo: t4c3e3

La función `kill()` `<signal.h>`

- Hasta aquí solo hemos visto señales enviadas por el kernel
- `kill()` permite enviar señales de un proceso a otro, conociendo su PID
- Comando kill: `kill [-s señal|-p][-l]pid`

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

pid > 0 la señal sig es enviada al proceso con identificador PID
pid = -1 , si el proceso tiene permisos de supervisor, enviaría una señal a todos los procesos
pid < -1 la señal sig es enviada a todos los procesos cuyo número de grupo coincide con el valor absoluto del pid especificado

Ver ejemplo: t4c3e4 Ver ejemplo: t4c3e5

Otras funciones

- Para enviar señales a un grupo de procesos
  - Función killpg()

```
int killpg (int pgrp, int sig);
```

- Para que un proceso se mande señales a sí mismo
  - Función raise()

```
int raise (int sig);
```

La función `alarm()`

`<unistd.h>`

- La función `alarm()` permite establecer un temporizador en el sistema de manera que su acción termine en un tiempo especificado en el futuro
- Al crear el temporizador se define un tiempo que el temporizador consume; **cuando el temporizador llega a cero, se genera la señal SIGALRM**

```
unsigned int alarm (unsigned int segundos);
```

Solo puede existir una alarma por proceso, si se invoca más de una vez, se actualiza el temporizador

Ver ejemplo: t4c3e6

Bloqueo y suspensión de señales…

- Se considera que una señal ha sido **entregada a un proceso** cuando **éste realiza la acción correspondiente** asociada a esa señal. Durante el tiempo que transcurre desde que la señal es generada hasta su entrega a un proceso se dice que la señal está **pendiente.**
- Los procesos tienen la opción de **bloquear** (o sea, demorar) la recepción de una determinada señal
- Si el programador **bloquea una señal** cuya acción asociada es la ejecución de una rutina propia o bien su tratamiento por defecto, **la señal queda pendiente.**
- Al descubrir que la señal **está bloqueada**, el kernel guarda la señal hasta que el proceso
  - desbloquee la señal, o
  - modifique la acción asociada a ésta y pase a ignorar esa señal

Funciones asociadas <signal.h>

```
int sigemptyset (sigset_t *set);

int sigfillset (sigset_t *set);

int sigaddset (sigset_t *set, int signum);

int sigdelset (sigset_t *set, int signum);

int sigismember (const sigset_t *set, int signum);
```

Ver ejemplo: t4c3e7

Bloqueo de señales `<signal.h>`

- El hecho de crear un conjunto de señales y luego agregar o eliminar señales del mismo, sin embargo, no significa la creación de un **handler de señales,** ni permite interceptar o bloquear señales
- Para establecer, modificar o interrogar por la máscara de señales que se desea bloquear se utilizará la función `sigprocmask()`
- Una vez que se establezca una máscara de señal, se deberá registrar un handler para la señal o señales que se desea interceptar, por medio de `signal()` o de `sigaction()`

Bloqueo de señales <signal.h>

- Función `sigprocmask()`

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

how	Efecto
SIG_BLOCK	El conjunto de señales bloqueadas es la unión del actual y el del argumento *set*
SIG_UNBLOCK	Las señales incluidas en *set* son eliminadas del conjunto de señales bloqueadas en la actualidad. Se considera permitido el intento de desbloquear una señal que no está bloqueada.
SIG_SETMASK	El conjunto de señales bloqueadas pasa a ser el contenido en el argumento *set*

Ver ejemplo: t4c3e8

Detección de señales pendientes

<signal.h>

- La función `sigpending()` hace posible que un proceso detecte señales pendientes (señales que fueron generadas mientras las mismas se encontraban bloqueadas) y luego decidir si ignorarlas o permitir que lleven a cabo su acción
- Por ejemplo, supongamos que se desea grabar datos en un fichero y no queremos que dicha operación sea interrumpida; así pues, antes de comenzar el proceso de escritura se bloquearán SIGTERM y SIGQUIT. Una vez terminada la grabación de los datos se podría verificar la existencia de señales pendientes y si una de las anteriores se encontrara pendiente se procedería a su desbloqueo. O, sencillamente, se las podrá desbloquear sin preocuparse de verificar previamente si está o no pendiente
- Función `sigpending ()`

```
int sigpending (sigst_t *set);
```

Ver ejemplo: t4c3e9