Práctica 6. Señales

1.- Objetivos

- Utilizar los mecanimos que proporciona el sistema operativo *UNIX* para que un proceso detecte la ocurrencia de un cierto evento.
- Trabajar con temporizadores.

2.- Operaciones con señales: *kill*, *signal* y *pause*.

Las señales son interrupciones software que pueden ser enviadas a un proceso para informarle de alguna situación especial.

Cuando un proceso recibe una señal puede actuar de tres formas distintas:

1. Ignorar la señal.
2. Invocar a la rutina de tratamiento por defecto, que no es codificada por el programador, sino que viene incorporada en el núcleo (*kernel*) del sistema operativo.
3. Invocar a una rutina de tratamiento definida por el usuario. En este caso es el propio programador quien debe codificar dicha rutina y el programa no va a terminar salvo que así lo indique la rutina de tratamiento.

Cada señal tiene asociado un número entero positivo. Dicho valor es recibido por el proceso destinatario de una señal.

En el archivo de cabecera *signal.h* se define una constante asociada a cada una de las diferentes señales que puede manejar el sistema. Así, cuando se incluye este archivo de cabecera, se pueden utilizar los nombres de las señales en lugar de sus valores numéricos. Los nombres y los valores de las diferentes señales pueden encontrarse en la ayuda de *signal*. Las señales que pueden ser manejadas por el sistema, junto con sus nombres, se definen en el fichero de cabecera *linux/signal.h*, que a su vez se halla incluido en *signal.h*.

El listado completo de señales disponibles puede visualizarse ejecutando el comando *kill -l* desde la *shell*:

```
 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
```

La siguiente tabla explica algunas de las señales más comunes:

Señal	Valor	Acción	Observaciones
SIGHUP	1	Term	Cuelgue detectado en la terminal de control o muerte

			del proceso de control.
SIGINT	2	Term	Interrupción procedente del teclado.
SIGQUIT	3	Core	Terminación procedente del teclado.
SIGILL	4	Core	Instrucción ilegal.
SIGABRT	6	Core	Señal de aborto procedente de *abort.*
SIGFPE	8	Core	Excepción de coma flotante.
SIGKILL	9	Term	Señal de matar.
SIGSEGV	11	Core	Referencia inválida a memoria.
SIGPIPE	13	Term	Tubería rota: escritura sin lectores.
SIGALRM	14	Term	Señal de alarma de *alarm.*
SIGTERM	15	Term	Señal de terminación.
SIGUSR1	10	Term	Señal definida por usuario 1.
SIGUSR2	12	Term	Señal definida por usuario 2.
SIGCHLD	17	Ign	Proceso hijo terminado o parado.
SIGCONT	18		Continuar si estaba parado.
SIGSTOP	19	Stop	Parar proceso.
SIGTSTP	20	Stop	Parada escrita en la *tty.*
SIGTTIN	21	Stop	E. de la *tty* para un *proc*. de fondo.
SIGTTOU	22	Stop	S. a la *tty* para un *proc.* de fondo.

Las entradas de la columna *Acción* de la tabla especifican la acción por defecto para la señal de la siguiente manera:

- *Term*: La acción por defecto es terminar el proceso.
- *Ign*: La acción por defecto es ignorar la señal.
- *Core:* La acción por defecto es terminar el proceso y realizar un volcado de memoria.
- *Stop:* La acción por defecto es detener el proceso.

Envío de señales: *kill*.

Para enviar señales a un proceso se utiliza la llamada *kill*:

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

`pid` es el identificador del proceso al que se envía la señal.

`sig` es el número de la señal que se quiere enviar o bien su nombre.

Si la señal se envía correctamente, *kill* devuelve *0*; en caso contrario devuelve *-1* y en *errno* estará el código del error producido.

Así, por ejemplo, para enviar la señal número *25* a un proceso cuyo *pid* esté almacenado en la variable `mipid`, se podría utilizar el siguiente código:

```
#include <signal.h>
…
pid_t mipid;
int a;
…
a=kill(mipid, SIGCONT);
if (a!=0) {
   perror("Llamada a kill.");
   exit(-1);
}
```

También es correcto efectuar la llamada utilizando los valores numéricos de las señales, por lo que en el ejemplo anterior también se podría utilizar:

```
a=kill(mipid, 25);
```

Tratamiento de señales: *signal*.

Cuando un proceso recibe una señal determinada ejecuta la acción por defecto asociada a esa señal, que es la que aparece en la tabla 7.1.

Cuando se desea modificar el comportamiento tras la recepción de una señal de manera que no se ejecute la acción por defecto, sino otra función diferente definida por el usuario, se utiliza la llamada *signal*:

```
#include <signal.h>
int (*signal(int sig, void(*action)()))();
```

`sig` es el número de la señal en cuestión.

`action` es la acción que se quiere iniciar la siguiente vez que se reciba la señal `sig`. Puede tener tres tipos de valores:

1. *SIG_DFL*: Ejecutar la acción por defecto.
2. *SIG_IGN*: Ignorar la señal, es decir, no ejecutar ninguna acción en absoluto.
3. Nombre de la función del tipo *void handler(int sig);* La función *handler* es la que se va a ejecutar la próxima vez que se reciba la señal y recibirá como parámetro el valor de la misma, *sig*.
   La llamada a la rutina *handler* es asíncrona, es decir, puede tener ligar en cualquier instante de la ejecución del programa.

Si se produce algún error, *signal* devuelve el valor *SIG_ERR* y en *errno* estará el código del error producido.

Observe que la acción que se fegine con *signal* solamente es válida para la próxima vez que se reciba la señal especificada. La siguiente vez que se ejecutará nuevamente la acción por defecto. Entonces, si se desea que siempre se ejecute la rutina *handler* es necesario indicarlo en al código de la propia función *handler* y al principio del mismo. Tal como se indica en el siguiente ejemplo:

```
…
void muestraNumero(int sig); // Función de tipo handler
int main(int argc, char* argv[]) {
   …
   /* Se le asigna el handler a la señal SIGUSR1 */
   if (signal(SIGUSR1, muestraNumero)==SIG_ERR) {
      perror("Llamada a signal.");
      exit(-1);
   }
   /* La proxima vez que se reciba la señal SIGUSR1
    * se llamará a la función muestraNumero()
    */
```

```
   …
}
void muestraNumero(int sig) {
   /* Se vuelve a asignar el handler a la señal SIGUSR1 */
   if (signal(SIGUSR1, muestraNumero)==SIG_ERR) {
      perror("Llamada a signal.");
      exit(-1);
   }
   /* Se ejecuta el código de la rutina de tratamiento */
   printf("Número de señal recibida: %d", sig);
}
```

Tratamiento de señales: *sigaction*

Signal no es la única forma de armar y tratar una señal, otra posibilidad es utilizar la llamada al sistema *sigaction:*

```
#include <signal.h>
int sigaction(int sig, struct sigaction *act, struct sigaction *oact);
```

Características:

- La llamada devolverá -1 si falla.
- Hay que decirle como primer parámetro la señal que se va a tratar.
- El segundo y tercer parámetro es un puntero a una estructura de tipo sigaction que determina a cuál va a ser el nuevo comportamiento (el primer puntero) y cuál era el antiguo (el segundo puntero.

La estructura *sigaction* se encuentra definida en el fichero signal.h y está formada por los siguientes campos:

```
struct sigaction {
      void (*sa_handler)();   /* Manejador para la señal */
      sigset_s sa_mask;       /* Señales bloqueadas durante la
                              ejecución del manejador */
      int sa_flags            /* Opciones especiales */
}
```

El primer argumento indica la acción a ejecutar cuando se reciba la señal y puede recibir los mismos valores que recibe *signal* para el parámetro *action* que veíamos en la sección anterior.
El segundo parámetro se refiere a las señales que queremos que se bloqueen mientras se está ejecutando la función manejadora. Por defecto solo se impide la recepción de la misma señal que provocó la llamada a la manejadora. El tercer parámetros es un campo de bits que especifica alguna de las siguientes opciones:

- SA_NOCLDSTOP: no queremos que se genere SIGCHLD cuando un hijo es parado.
- SA_RESETHAND: cuando se vuelva de la función manejadora, restaurar el comportamiento de la señal al comportamiento por defecto.
- SA_RESTART: hacer que las llamadas al sistema no devuelvan -1 y den el error EINTR cuando se produzca la señal.
- SA_NODEFER: hacer que la señal no se bloquee dentro de la función manejadora como

ocurre por defecto.

A continuación se muestra un ejemplo del uso de esta llamada.

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

void nonada(int s)
   {
    printf("PADRE: Estoy en la manejadora.\n");
    }

int main(void)
   {
    char lInea[40];
    int valor_devuelto;
    struct sigaction ss;

    switch (fork())
       {
        case -1: perror("fork"); return 1;
        case 0: /* HIJO */
            for (;;) pause();
        default: /* PADRE */
            ss.sa_handler=nonada;
            if (-1==sigaction(SIGUSR1,&ss,NULL))
               {perror("PADRE: sigaction");
                return 1;}

            printf("PADRE: SIGUSR1(%d) registrada.\n",SIGUSR1);

            if (wait(&valor_devuelto)==-1)
               {perror("PADRE: wait");
                return 1;}
        }

    return 0;
    }
```

Como se puede ver tanto *signal* como *sigaction* permiten el tratamiento y bloqueo de señales. *Sigaction* forma parte del estándar POSIX de llamadas al sistema mientras que *signal* se encuentra definida en el estándar C. Algunas diferencias:

- *Sigaction* es más completo que *signal* y permite configurar más parámetros.
- La función *signal* no bloquea la llegada de otras señales mientras se está ejecutando la función manejadora mientras que *sigaction* puede bloquear otras señales hasta que la manejadora finaliza.
- La señal *signal* resetea el comportamiento para la señal al valor por defecto una vez se recibe la señal. Esto significa que lo primero que debe aparecer en la manejadora es de nuevo el bloqueo de la señal mediante el uso de *signal*, eso supone que sea posible que llegue cualquier otra señal o esa entre el tiempo que se recibe la señal y que se puede rearmar dentro de la manejadora.

En espera de señales: *pause*.

Cuando interesa que un proceso suspenda su ejecución en espera de que ocurra algún evento exterior a él, se utiliza la llamada a *pause*:

```
#include <unistd.h>
int pause(void);
```

3.- Empleo de temporizadores

En ciertas aplicaciones puede ser interesante que el código se ejecute de acuerdo con una temporización. Esto puede venir impuesto por las especificaciones del programa en situaciones en las que una respuesta antes de tiempo pueda ser tan perjudicial como una con retraso.

La llamada al sistema que resuelve este problema es *alarm*:

```
#include <unistd.h>
unsigned alarm(unsigned sec);
```

alarm activa un temporizador que inicialmente toma el valor `sec` segundos y que se va decrementando en tiempo real. Una vez transcurrido ese tiempo, el proceso recibirá la señal *SIGALRM* (valor *14*) cuya acción por defecto será finalizar el proceso.

4.- Desarrollo de la práctica

Señales del sistema

Modifica el código de la práctica anterior para que no atienda a la señal producida por la pulsación de *Control+C* (*SIGINT*).

Sincronización de señales

Dado el siguiente programa:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define M 5.0 // Duración de la tarea del proceso padre
#define N 7.0 // Duración de la tarea del proceso padre

int main(void){
   int x, i=0;
   pid_t pid;

   pid=fork();
   if (pid==-1){
      perror("Error en la llamada a fork()");
      exit(0);
   } else if (pid == 0){
      while(1){
```

```
         srand (getpid());
         x=1+(int)(N*rand()/ RAND_MAX +1.0);
         /* X es un número aleatorio entre 1 y N */
         printf("COMIENZO TAREA HIJO %d\n",i);
         sleep(x);
         printf("FIN TAREA HIJO %d\n",i);
         i++;
      }
   } else {
      while(1) {
         srand (getpid());
         x=1+(int)(M*rand()/ RAND_MAX +1.0);
         /* X es un número aleatorio entre 1 y N */
         printf("COMIENZO TAREA PADRE %d\n",i);
         sleep(x);
         printf("FIN TAREA PADRE %d\n",i);
         i++;
      }
   }

   return 0;
}
```

Sincroniza los procesos hijo y padre de tal manera que no se solapen las tares del hijo y del padre.

Señales periódicas

Realiza un programa que muestre la hora del sistema por pantalla cada 10 segundos. Utilice la función *alarm()*.