Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

PRÁCTICA 2. COMUNICACIÓN BÁSICA ENTRE PROCESOS - SEÑALES

1. OBJETIVO

Se pretende mostrar al alumno mecanismos simples de comunicación entre procesos. Para ello se van a estudiar las señales.

2. DESCRIPCIÓN

Señales:

Las señales son una forma de notificar al proceso acerca de la ocurrencia de un evento o error. Un proceso se comporta frente a una señal como el procesador frente a una interrupción, por eso se dice que las señales son interrupciones software. El origen de una señal es el sistema operativo u otro proceso. El comportamiento frente a la señal es de la siguiente forma:

- El proceso detiene su ejecución en la instrucción máquina que está ejecutando.
- Pasa a ejecutar una rutina de tratamiento de la señal (manejador de la señal), cuyo código forma parte del propio proceso.
- Una vez terminada la ejecución de esta rutina, sigue la ejecución por la instrucción donde fue interrumpido.

La lista de señales posibles se pueden encontrar en el fichero de cabecera `signal.h`. A continuación se ponen algunos ejemplos de señales:

```
SIGABRT: terminación anormal.
SIGALRM: señal de fin de temporización.
SIGFPE: operación aritmética errónea.
SIGINT: señal de interrupción.
SIGKILL: señal de terminación.
SIGSEG: señal de referencia a memoria inválida.
```

Las señales proporcionan el mecanismo más básico de comunicación entre procesos.

Frente a las señales, el proceso puede:

- Ignorarlas: la señal se desecha y no se tiene en cuenta. Hay señales que no pueden ser ignoradas por el proceso, por ejemplo SIGKILL.
- Bloquearlas: la señal no se tiene en cuenta, pero queda pendiente hasta que se desbloquea.
- Capturarlas: cuando llega la señal el proceso ejecuta una rutina de manejo de señal. El proceso puede indicar el manejador de señal que será ejecutado cuando se recibe la señal,

esto se conoce como armar una señal. Hay señales que no pueden ser armadas por el proceso, por ejemplo SIGKILL.

Existen servicios para tratar conjuntos de señales, para el envío de señales, para indicar el manejador de una cierta señal, para el bloqueo y desbloqueo de señales, para esperar por la recepción de una señal y para la gestión de temporizadores. De todas estas funciones, se van a estudiar las siguientes:

```
int sigemptyset(sigset_t *set);
```

Inicia un conjunto de señales de modo que no contenga ninguna señal.

```
int sigaddset(sigset_t *set, int signo);
```

Añade una señal (signo) al conjunto de señales previamente iniciado (set).

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

Envía la señal sig al proceso o grupo de procesos especificado por pid. Si pid es mayor que cero, la señal se enviará al proceso con identificador de proceso igual a pid. Si pid es cero, la señal se enviará a todos los procesos cuyo identificador de grupo sea igual al identificador de grupo del proceso que envía la señal. Si pid es negativo pero distinto de –1, la señal se enviará a todos los procesos cuyo identificador de grupo sea igual al valor absoluto de pid. Para pid igual a –1 no se especifica nada.

Sólo se puede enviar una señal a un proceso con el mismo identificador de usuario real o efectivo, a no ser que el proceso que envía tenga los privilegios adecuados (por ejemplo, es un proceso del superusuario).

```
int sigaction(int sig, struct sigaction *act, struct sigaction *oact);
```

Permite indicar la función que manejará la señal cuando sea recibida. Tiene tres parámetros. El primer parámetro (sig) indica el número de señal para la cual se establecerá el manejador. El segundo parámetro (act) es un puntero a una estructura del tipo sigaction para establecer el nuevo manejador. El último parámetro (oact) es un puntero a una estructura del mismo tipo donde se almacena la información sobre el manejador asociado a la señal anteriormente.

La estructura sigaction está definida de la siguiente forma:

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

El primer campo (sa_handler) indica la acción a realizar. Su valor puede ser:

- SIG_DFL: acción por defecto, en la mayoría de los casos consiste en matar al proceso.
- SIG_IGN: ignora la señal.
- Una función que devuelve void y que acepta como parámetro un número entero. Este parámetro es el número de la señal que se ha enviado. Esto permite utilizar un mismo manejador para distintas señales.

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

Espera la recepción de una señal. Bloquea al proceso que la invoca hasta que llegue la señal.

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

Activa un temporizador. Envía al proceso la señal SIGALRM después de pasados el número de segundos especificados en el parámetro seconds.

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

Suspende un proceso durante un número determinado de segundos.

⇒ **Ejemplo:** El siguiente programa hace que el proceso ignore la señal SIGINT que se genera cuando se pulsa CTLR-C. Luego ejecuta un bucle infinito. Para terminarlo, se puede parar y luego enviar una señal SIGKILL con la orden kill (kill –SIGKILL pid).

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

int main(void)
{
   struct sigaction act;

   act.sa_handler= SIG_IGN;    /* Ignora la señal */
   act.sa_flags= 0;            /* Ninguna accion especial */
   sigemptyset(&act.sa_mask); /* Ninguna señal bloqueada */

   sigaction(SIGINT, &act, NULL);

   while(1);                   /* Bucle infinito */

   return 0;
}
```

p2_prueba1.c

⇒ **Ejemplo de alarma:**

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

void tratar_alarma(void)
{
   printf("Activada \n");
}

int main(void)
{
   struct sigaction act;

   act.sa_handler = tratar_alarma; /* función a ejecutar */
   act.sa_flags = 0;               /* ninguna acción especifica */
   /* Se bloquea la señal SIGINT cuando se ejecute la función
      tratar_alarma */
   sigemptyset(&act.sa_mask);
```

```
   sigaddset(&act.sa_mask, SIGINT);

   sigaction(SIGALRM, &act, NULL);

   for(;;)
   {
      alarm(3); /* se indica al SO que envie una señal ALRM a los 3 seg */
      pause();  /* se suspende el proceso hasta que se reciba una señal */
   }

   return 0;
}
```

p2_prueba2.c

⇒ Ejemplo de kill:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <wait.h>
pid_t pid;
void matar_proceso(void)
{
   kill(pid, SIGKILL); /* se envía la señal al hijo */
}


int main(int argc, char **argv)
{
   int status;
   struct sigaction act;

   /* Se crea el proceso hijo */
   pid = fork();

   switch(pid)
   {
      case -1:   /* error del fork() */
         perror("fork");
      break;
      case 0:    /* proceso hijo */
         if(argc>1)
         {  /* El proceso hijo ejecuta el mandato recibido */
            execvp(argv[1], &argv[1]);
            perror("exec");
         }
      break;
      default:   /* padre */
      /* establece el manejador */
         act.sa_handler = matar_proceso; /*función a ejecutar*/
         act.sa_flags = 0;   /* ninguna acción especifica */
         sigemptyset(&act.sa_mask);

         sigaction(SIGALRM, &act, NULL);

         alarm(5);

         /* Espera al proceso hijo */
         wait(&status);
   }
```

```
    return 0;
}
```

p2_prueba3.c

TAREAS:

1. Descarga el fichero combasica_signals.zip correspondiente a la segunda práctica.
2. Compila y ejecuta el programa p2_prueba1.c. Comprueba que al pulsar CTLR-C no termina. Para el proceso (CTLR-Z) y termínalo con la orden kill (kill -SIGKILL pid).
3. Compila y ejecuta el programa p2_prueba2.c. Indica qué es lo que hace el programa.
4. Compila el programa p2_prueba3.c. Ejecuta el programa con el programa p2_prueba2 como argumento. ¿Cuál es el resultado?
5. Realiza un ejemplo básico de productor/consumidor que se comuniquen y sincronicen mediante señales en un fichero llamado prod_cons_sig.c. Para ello:

- ♦ Crea un proceso hijo con fork. El padre será el productor y el hijo el consumidor.
- ♦ El productor cada 5 segundos (sleep(5)) enviará una señal al consumidor empezando por la señal 1 y terminando con la 5.
- ♦ El consumidor debe quedarse indefinidamente a la espera, capturar estas señales e imprimir su valor por pantalla mediante la orden printf("Recibida %d\n",sig); donde sig es la señal recibida.
- ♦ Para terminar, el productor transcurridos 5 segundos (sleep(5)) desde la última señal enviada manda una señal SIGKILL al consumidor.

6. Envía en la tarea habilitada para la práctica 2 un fichero practica2.txt con las tareas comentadas y el fichero prod_cons_sig.c del apartado 5.