Apartado A. Gestión de procesos y Señales

Enunciado

La práctica constará de dos ficheros fuente: `padre.c` y `hijo.c`. El programa padre no admitirá nada como argumentos de la línea de órdenes de la *shell*. Hará lo siguiente:

- Tendrá dos hijos. Uno de ellos escribirá en la pantalla "`Soy el primer hijo (H1) y mi PID es <número_de_pid>`" y no acabará pero tampoco hará nada más.
- El otro hijo escribirá en la pantalla "`Soy el segundo hijo (H2) y mi PID es <número_de_pid>`" y pasará a ejecutar el código resultante de la compilación de `hijo.c`.
- El padre se quedará esperando por la finalización de cualquiera de sus hijos. Cuando reciba la información de la muerte del primer hijo (H1), comprobará que ha muerto por una señal e imprimirá el número de dicha señal que le mató: "`El primer hijo ha muerto al recibir la señal número <número_de_señal>`". A continuación, enviará una señal `SIGUSR1` a su segundo hijo (H2). Si recibe la información de la muerte del segundo hijo (H2), comprobará que es una terminación normal e imprimirá el código de retorno que le devuelve H2: "`El segundo hijo ha muerto y su código de retorno es <código_de_retorno>`". Acto seguido finalizará normalmente y devolverá como código de retorno un cero.

En `hijo.c`, lo primero que se hace es dormir cinco segundos (usad `sleep`), matar a su hermano H1 con la señal `SIGTERM` y no acabar. Cuando se reciba la señal `SIGUSR1`, se acaba con un código de retorno igual a 1.

Si el esquema que se sigue no es el previsto por cualquier circunstancia, el padre devolverá 1. Se permitirá pasar un argumento al programa `hijo`. Mientras están esperando, los procesos involucrados no deben consumir CPU (debe utilizarse la llamada al sistema `pause`).

Observaciones

En esta práctica debéis tener cuidado con un problema de sincronización. No quiere decir que siempre se tenga este problema, pero puede darse el caso. El problema puede originarse cuando un proceso tiene que interceptar una señal, y para evitarlo, se tiene que garantizar que:

- La señal nunca llegará ANTES de que se haya instalado el manejador correspondiente (mediante la invocación a `sigaction`).
- SI se ha realizado una pausa (mediante la llamada al sistema `pause`, que permite suspender un proceso hasta recibir una señal, de modo que el proceso no consuma CPU), Y se quiere que el proceso continúe después de la pausa (cuando se reciba la señal), ENTONCES se debe garantizar que el proceso realiza la pausa ANTES de recibir la señal, pues si no, se quedaría atrapado en ella para siempre.

Apartado B. Hilos de ejecución

Dadas las definiciones de las siguientes funciones: `hilo1` e `hilo2`,

```
#include <stdlib.h>
#include <stdio.h>

int x ;
```

```
void hilo1() {
  int i,j;
  char c;

  c=getchar();
  x=atoi(c);/*convierte a entero*/
  j=5;
  for (i=1; i<=10; i++) {
    j = j + x;
    x = x + 1;
  }
  printf("j:%d\n", j );
}
```

```
void hilo2() {
  int k, j;

  k = x * 2;
  for(j=1; j<=3; j++) {
    k = k + j*x;
    x = x - 1;
  }
  printf( "k:%d \n", k );
}
```

Se pide escribir un programa, denominado `hilos.c`, que:

- utilizando hilos, ejecute concurrentemente las dos funciones dentro del proceso.
- utilizando semáforos, los accesos sobre la variable global `x` se realicen en exclusión mutua y el valor que ha de imprimir la función `hilo2` siempre aparezca en pantalla tras haberse escrito el del `hilo1`.