

Ejercicio 3:

a) Analiza el texto que imprime el programa. ¿Se puede saber a priori en qué orden se imprimirá el texto ? ¿Por qué? (0.5 punto)

No podemos saber como se va a imprimir todo el programa, ya que al crearse en un primer momento el proceso hijo, este y el padre suceden de igual manera, lo que hace que no conozcamos cual de estos se va a imprimir primero.

b) Cambia el código para que el proceso hijo imprima su pid y el de su padre en vez de la variable i. (0.5 puntos)

Para conseguir esto hay que modificar la línea en la que imprime el hijo de la siguiente manera:

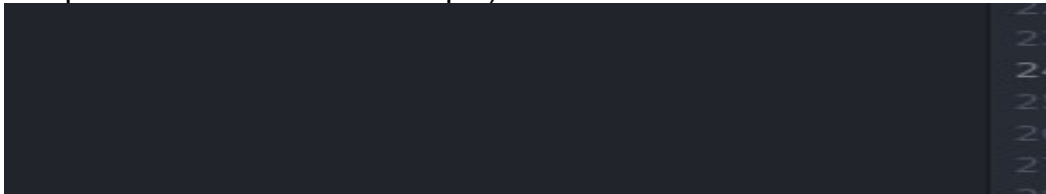
```
printf("HIJO %d --> PADRE %d\n", getpid(), getppid());
```

c) Analiza el árbol de procesos que genera el código de arriba. Muestralo en la memoria como un diagrama de árbol (como el que aparece en el ejercicio 4) explicando porqué es así. (0.5 punto)

```
Padre --> Hijo 1
        --> Hijo 2
        --> Hijo 3
```

a) El código del ejercicio 3 deja procesos huérfanos, ¿Por qué?. (0.25 puntos)

En un principio no se aprecia. Si ejecutamos el programa con normalidad los procesos hijos terminan antes que el padre como podemos ver en la siguiente ejecución (en este programa el padre muestra también su pid):



Pero si ponemos un sleep de 10 segundos antes de imprimir el pid del proceso padre de los hijos, podemos ver que deja huérfanos a dos procesos. Siendo esto debido a que no hace un wait por cada proceso hijo que ha creado.

```
testing test3
PADRE 6255
PADRE 6255
PADRE 6255
HIJO 6258 --> PADRE 6255
HIJO 6256 --> PADRE 6255
HIJO 6257 --> PADRE 1
```

Nota: que solo haya un proceso huérfano no implica que no puedan haber dos. Nunca tres porque hace un wait

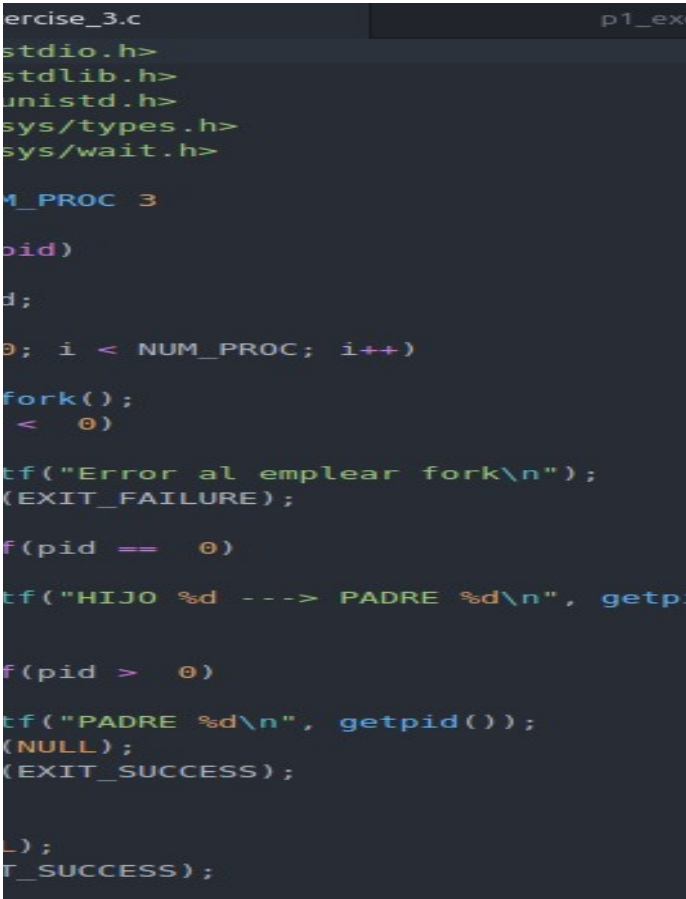
b) Introduce el mínimo número de cambios en el código del ejercicio 3 para que no deje procesos huérfanos. (0.25 punto).

Para solucionar esto hemos añadido un bucle en el wait final para que realice el wait las mismas veces que procesos hijos ha creado.

```
for(i=0;i<NUM_PROC;i++)  
    wait(NULL);
```

c) Escribe un programa en C (ejercicio4.c) que genere el siguiente árbol de procesos. El proceso padre genera un proceso hijo que a su vez generará otro hijo así hasta llegar a NUM_PROG procesos hijos. Asegurate de que cada padre espera a que termine su hijo y no queden procesos huérfanos. (1 puntos)

El código queda de la siguiente forma:



```
ercise_3.c p1_ex  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/wait.h>  
  
#define NUM_PROC 3  
  
int main()  
{  
    pid_t pid;  
    for(i=0; i < NUM_PROC; i++)  
    {  
        if(fork() < 0)  
        {  
            printf("Error al emplear fork\n");  
            return (EXIT_FAILURE);  
        }  
        if(pid == 0)  
        {  
            printf("HIJO %d ---> PADRE %d\n", getpid(), getppid());  
            // Child process logic  
        }  
        if(pid > 0)  
        {  
            printf("PADRE %d\n", getpid());  
            wait(NULL);  
        }  
    }  
    return (EXIT_SUCCESS);  
}
```

Ejercicio 5:

a) En el programa anterior se reserva memoria en el proceso padre y la inicializa en el proceso hijo usando el método strcpy (que copia un string a una posición de memoria), una vez el proceso hijo termina el padre lo imprime por pantalla. ¿Qué ocurre cuando ejecutamos el código? ¿Es este programa correcto? ¿Por qué? justifica tu respuesta. (0.5 puntos)

Se muestra por terminal "Padre: ". Por lo que deducimos que el programa es incorrecto.

No muestra lo deseado por pantalla porque cuando se crea un proceso nuevo con fork, este proceso es una copia idéntica del anterior, pero los cambios que le suceden al proceso hijo no afectan al proceso padre. Es por esto que no se imprime la cadena la cadena al tratar de imprimirla en el proceso padre, porque esta información está en el proceso hijo.

b)El programa anterior contiene un memory leak ya que el array sentence nunca se libera. Corrige el código para eliminar este memory leak.¿Dónde hay que liberar la memoria en el padre, en el hijo o en ambos?. Justifica tu respuesta. (0.5 puntos)

Tan solo es necesario liberar la memoria del padre, ya que el padre hace wait y cuando el proceso hijo termina siempre se libera su memoria y datos que era lo que no compartía con el padre.

Ejercicio 7:

A la hora de realizar este ejercicio y viendo el código proporcionado, la principal dificultad fue utilizar las nuevas funciones y entender los parámetros que hay que pasarlas:

execv. No tiene la 'p' por lo que en el primer argumento hay que especificar todo el path del comando a utilizar (/bin/cat). El resto se para todo como un array de cadenas de caracteres.

execlp. Tiene la 'p', por lo que solo pusimos como primer argumento "ls". El resto de argumentos son las especificaciones, cada una como un argumento propio.

Ejercicio 9:

A la hora de realizar este ejercicio hay que tener en cuenta dos cosas.

La primera es haber creado las tuberías antes de intentar pasar los datos (y comprobar que se han creado bien).

La segunda es que hay que para enviar se cierra el '0' y se escribe en el '1' y para leer al contrario.

Ejercicio 12:

En este ejercicio creamos una estructura para pasar el input y el output de las variables. Como utilizamos hilos para ejecutar esa estructura es muy importante que devuelva un void y que sus parámetros sean un void. Luego se realizarán los casteos necesarios para permitir que todo funcione.