Memoria práctica 1

Rodrigo Lardiés Guillén, Carlos Gómez-Lobo Hernaiz

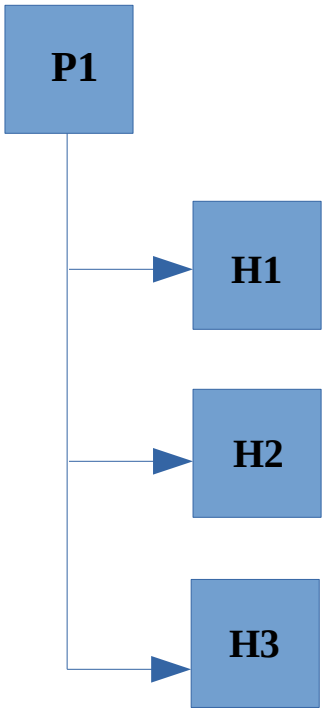
Ejercicio 3. a) Analiza el texto que imprime el programa. ¿Se puede saber a priori en qué orden se imprimirá el texto ? ¿Por qué?

No, al estar ejecutándose dos procesos en paralelo que mandan imprimir texto en la terminal en un espacio de tiempo muy pequeño, dependerá del planificador del sistema operativo cuál de los dos se ejecutará antes.

b) Cambia el código para que el proceso hijo imprima su pid y el de su padreen vez de la variable i.

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

#define NUM_PROC 3

int main(void){

	pid_t pid;
	int i;
	for(i = 0; i < NUM_PROC; i++) {

		pid = fork();
		if(pid <  0) {
			printf("Error al emplear fork\n");
			exit(EXIT_FAILURE);
		}
		else if(pid ==  0){
			printf("HIJO %d PADRE %d\n", getpid(), getppid());
			exit(EXIT_SUCCESS);
		}
		else if(pid >  0){
			printf("PADRE %d\n", i);
		}
	}
	wait(NULL);
	exit(EXIT_SUCCESS);
}
```

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.



En cada iteración del bucle *for*, el proceso padre crea un hijo que termina justo después mientras que el padre hace la siguiente iteración, repitiendo la iteración hasta que el bucle acabe y creando así tantos procesos hijos como valor tenga la macro *NUM_PROC*.

Ejercicio 4. a) El código del ejercicio 3 deja procesos huérfanos, ¿Por qué?

Porque al poner *wait(null)* el proceso padre solo espera a que muera el primer proceso, y entonces también se muere, dejando otro u otros procesos todavía ejecutándose. Esto se puede verse mejor si ponemos un *sleep(i)* justo antes de imprimir en pantalla el pid de los hijos.

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

Este problema podría solucionarse simplemente cambiando de sitio la instrucción *wait(null)*, poniéndola dentro del bucle, junto con la instrucción que hace que el padre imprima en pantalla. De esta forma el padre espera a que muera cada proceso, antes de crear el siguiente o morir, evitando así dejar procesos huérfanos.

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

#define NUM_PROC 3

int main(void)
{
	pid_t pid;
	int i;
	for(i = 0; i < NUM_PROC; i++)
	{
	pid = fork();
	if(pid <  0)
	{
		printf("Error al emplear fork\n");
		exit(EXIT_FAILURE);
	}
	else if(pid ==  0)
	{
		sleep(i);
		printf("HIJO %d PADRE %d\n", getpid(), getppid());
		exit(EXIT_SUCCESS);
	}
	else if(pid >  0)
	{
		printf("PADRE %d\n", i);
		wait(NULL);
	}
	}
	exit(EXIT_SUCCESS);
}
```

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 hijoy no queden procesos huérfanos.

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

#define NUM_PROC 3

int main(void)
{
        pid_t pid;
        int i, wstatus;

        for(i = 0; i < NUM_PROC; i++) {
        pid = fork();
        if(pid <  0)
        {
                printf("Error al emplear fork\n");
                exit(EXIT_FAILURE);
        }
        else if(pid >  0)
        {
                printf("HIJO %d PADRE %d\n", getpid(), getppid());
                waitpid(pid, &wstatus, 0);
                exit(EXIT_SUCCESS);
        }
        else if(pid ==  0)
        {
                printf("PROC %d\n", i);
        }
        }
        exit(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 stringa 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.

Lo que ocurre es que el padre imprime *"PADRE:"*, pero no la frase. Esto se debe a que el proceso padre y el hijo no comparten memoria y, por tanto, aunque ambos tienen memoria reservada para la variable *sentence*, esta memoria es distinta y cuando el hijo cambia su valor, este no cambia para el padre, y por ello no imprime nada.

b) El programa anterior contiene un memory leakya que el array sentencenunca 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.

Para arreglar la pérdida de mamoria, sería necesario poner un *free* antes de que termine la ejecución de ambos procesos, uno para cada uno. Esto es así poque, como se explica anteriormente, ambos programas tienen memorias independientes, que por tanto tienen que liberarse de manera independiente.