Practica 1 SOPER

Juan Riera Gomez
juanriera3739@estudiante.uam.es
Carlos Ignacio Isasa
carlos.isasa@estudiante.uam.es
Pareja 21

Todos los fuentes se compilan automáticamente al ejecutar make.

Ejercicio 4

Análisis del árbol de procesos asociado al código. Modifica el código anterior de forma que cada hijo imprima su pid y el pid de su proceso padre.

El bucle se ejecuta tres veces, así que en un principio tenemos un proceso, cuando se ejecuta el primer fork, pasamos a tener dos: un hijo y su padre, cuando se ejecuta el segundo cada uno de éstos tiene un hijo, y por tanto tenemos cuatro, en el último fork tenemos, por tanto, 8 procesos. Editando el código para que imprima el estado de los procesos tras cada fork vemos que empezamos con un padre y un hijo, dos padres y dos hijos, cuatro padres y cuatro hijos:

```
e336799@16-24-64-246:~/workspace$ ./a
PADRE
HIJO
PADRE
PADRE
HIJO
HIJO
PADRE
PADRE
PADRE
PADRE
HIJO
HIJO
HIJO
HIJO
```

Tras haber vuelto a editar el código lo hemos guardado en el fuente llamado ejercicio4a.c cuya salida fue la que sigue:

```
e336799@16-24-64-246:~/workspace$ ./a
id superpadre = 3763
HIJO: pid 3764, pid del padre 3763
HIJO: pid 3765, pid del padre 3763
HIJO: pid 3766, pid del padre 3763
HIJO: pid 3767, pid del padre 3764
HIJO: pid 3768, pid del padre 3764
HIJO: pid 3769, pid del padre 3765
HIJO: pid 3770, pid del padre 3767
```

Este código hace una serie de fork y, si el proceso es hijo, imprime su propio pid y el de su padre. Además al empezar se imprime el pid del “superpadre”, el padre de todos. Vemos que coincide con los resultados esperados y comentados más arriba.

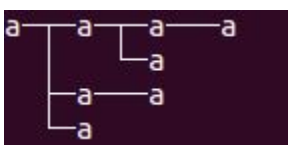
b) Explica la diferencia entre el código anterior y el siguiente:

El segundo código, al tener un wait añadido, cada proceso padre espera a que finalice la ejecución de todos sus procesos hijo, para recoger sus datos de salida y salir él. En el primer código esto no ocurría, dando lugar a proesos zombie.

¿Existen procesos huérfanos en alguno de los dos programas analizados? Al igual que en el código anterior, modifica este programa para que cada proceso hijo imprima su pid y el pid de su proceso padre. De cara a analizar la existencia de procesos huérfanos es de utilidad el comando pstree. Ejecuta pstree para todos pid de procesos padre que obtienes al ejecutar los dos programas de este ejercicio. Analiza la salida que obtienes para cada uno de los casos correspondientes.

Si, en el programa del primer enunciado, al no haber ninguna instrucción wait, los procesos padre no esperaban a que se ejecutarán los procesos hijo, y por tanto, éstos quedaban en estado zombie, quedando después huérfanos.

Aquí tenemos el árbol asociado al primer programa (que hemos obtenido añadiendo bucles infinitos en los procesos hoja, para que no se termine la ejecución, y poder ejecutar pstree):



Vemos que todos tienen un padre dado que hemos detenido la ejecución con un bucle y ninguno ha terminado, pero si un padre terminara, no esperaría a los hijos, y éstos

se quedarían huérfanos. Si terminaran los hijos, nadie tomaría su salida y se quedarían en estado zombie.

Código fuente editado del segundo programa para que tenga la funcionalidad pedida:

ejercicio4b.c

Ejercicio 5

a) Introduce el mínimo número de cambios en el código del segundo programa del ejercicio de forma que se generen un conjunto de procesos de modo secuencial (cada proceso tiene un único hijo y ha de esperar a que concluya la ejecución de su proceso hijo). Todos lo cambios introducidos han de explicarse adecuadamente.

El cambio que hemos hecho ha sido cambiar las instrucciones wait y exit de sitio. Ahora en vez de estar al final del programa están dentro del bucle y solo se ejecutan si el proceso es padre, de esta forma, si el proceso es hijo, seguirá iterando, y si es padre, esperará a su hijo y saldrá. Fuente:

ejercicio5a.c

b) Introduce el mínimo número de cambios en el código del segundo programa del ejercicio anterior de forma que exista un único proceso padre que dar lugar a un conjunto de procesos hijo. El proceso padre ha de esperar a que termine la ejecución de todos sus procesos hijo. Todos los cambios introducidos han de explicarse convenientemente.

En este caso, si el proceso es el proceso padre seguirá iterando y teniendo más hijos, y los procesos hijo saldrán del bucle (con break). Por último, dado que wait() solo espera a un hijo, la instrucción while(wait(&status)>0) {} hace que el proceso espere a todos sus hijos hasta que ya no tenga. Fuente:

ejercicio5b.c

Ejercicio 6

Escribe un programa en C (ejercicio6.c) que reserve en el proceso padre memoria dinámica para una cadena de 80 caracteres de longitud y después genere un proceso hijo. Si en el proceso hijo se pide al usuario que introduzca un nombre para guardar en la cadena, ¿el proceso padre tiene acceso a ese valor? ¿Dónde hay que l liberar la memoria reservada y por qué?

Para comprobar esto nuestro programa, además de dividirse en un fork, si es hijo, almacena una cadena pedida por línea de comandos en la variable ‘cadena’ y un valor distinto si es padre, y después los imprime. Con este experimento comprobamos dos cosas:

-El padre no tiene acceso a la variable ‘cadena’ de su hijo, luego al hacer el fork, la variable ‘cadena’ no apunta a la misma dirección en el proceso padre y en el proceso hijo. Por tanto si el proceso hijo pidiera un valor para almacenar en ‘cadena’ al usuario, el padre no tendría acceso a él.

-Además de apuntar a direcciones distintas, las dos direcciones tienen memoria reservada. Por tanto habría que liberar memoria en los dos procesos.

Ejercicio 8

Escribe un programa en C (ejercicio8.c) que cree tantos procesos hijo como el proceso padre reciba como argumentos de entrada, y que serán cualquier tipo de programa ejecutable. Cada proceso hijo mediante, la llamada a una función exec, debe ejecutar cada programa pasado por argumento.
Para saber que función exec se debe ejecutar, se pasará una única opción que lo indicará, de la siguiente forma:
-l: se ejecutará la función execl
-lp: se ejecutará la función execlp
-v: se ejecutará la función execv
-vp: se ejecutará la función execvp

Este ejercicio recibirá unos argumentos de entrada que leerá. Dependiendo del flag, que estará en la última posición, ejecutará los programas de una forma u otra. Lo que hace el programa es crear tantos hijos como programas sea necesario ejecutar (argc - 2) y según el flag ejecutarlos, comprobando que flag es con un strcmp.
Fuente: ejercicio8.c

Ejercicio 9

Dado que no se especifica en el enunciado, nuestro programa le pide al usuario que escoja entre los siguientes dos modos, teniendo que introducir la letra correspondiente (a o b):

-Modo a: si quiere que se muestre solo una operación, se le pide que introduzca introduzca una operación aritmética, después ejecuta, con esos valores, todas las operaciones en procesos hijo, e imprime solo la operación pedida por el usuario.

- Modo b: si quiere que se muestren todas las operaciones sólo se le piden los dos argumentos a introducir.