¿Qué es un proceso hijo?

Un proceso hijo no es ni mas ni menos que un clon del proceso original (Padre). Para ello existe un servicio o instrucción dentro de C, llamada ***FORK().***

Cuando un proceso crea otro proceso mediante el servicio **FORK**, ambos continúan ejecutando en la siguiente sentencia al ***FORK***, pero existirán dos procesos ejecutando el mismo código en lugar de un único proceso, como sucedía antes de invocar el servicio **FORK**.

En el caso del proceso padre (el que ha creado el hijo), el valor devuelto por el servicio **FORK** será el **PID** del proceso hijo recién creado, mientras que en el caso del proceso hijo, el valor devuelto por el servicio FORK será **0**. Obviamente, en caso de producirse un error durante la llamada a ***FORK***, no se creará el proceso hijo y el valor devuelto por ***FORK*** será **-1**.

**¿Como se diferencian el padre del hijo?**

Recuerden que cuando un proceso se crea, existen 4 partes fundamentales: ***CODE, DATA; STACK y PCB***, en esta ultima la **PCB** *(Process Control Block)*, existen varios registros como el **PID** *(Process Identification)* y **PPID** *(Process Parent Identification)*, acá se observara la herencia del proceso o mejor dicho cuál es su antecesor.

**Vamos a ver como seria un ejemplo en seudocódigo**

La llamada al sistema FORK() crea un clone exacto del proceso que hace la llamada.

Pero, ¿qué significa esto? Experimentemos con un nuevo programa:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

Int main(int argc, char\* argv[])

{

write(1, "one\n", 4);

fork();

write(1, "fork\n", 5);

exit(0);

}

***SALIDA***

one

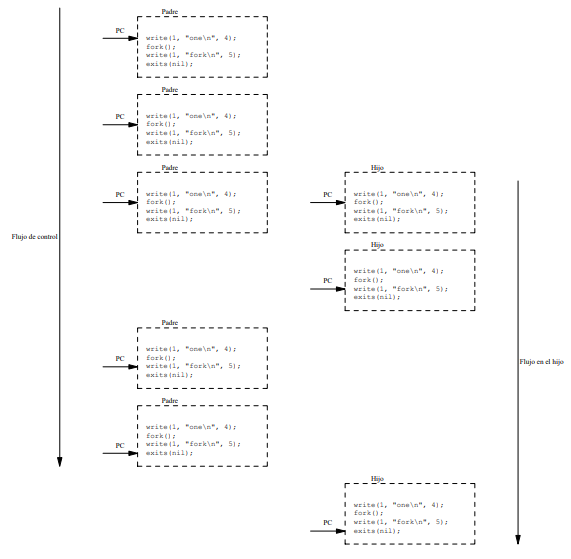
fork

fork

El primer **WRITE** ejecuta y vemos ***ONE*** en la salida. Pero después, llamamos a **FORK**, lo que crea otro proceso que es un clon exacto y, por tanto, están también dentro de la llamada a **FORK**. Ambos procesos (padre e hijo) continúan desde ese punto y, claro, llamarán al segundo write de nuestro programa.

Pero, naturalmente, los dos llaman a **WRITE**, por lo que vemos dos veces **FORK** en la salida del programa.

***Ejemplo en ejecucion***



Si seguimos la figura desde arriba hacia abajo vemos que inicialmente sólo existe el padre. Las flechas representan el contador de programa o **PC** *(Program Counter)* y vemos que el padre está ejecutando primero la primera llamada a **WRITE** del programa. Después, el padre llama a **FORK** y ¡aparece un nuevo proceso hijo! Cuando **FORK** termina su trabajo, tanto el proceso padre como el hijo están retornando de la llamada a **FORK**. Esto es lógico si piensas que el hijo es una copia exacta del padre en el punto en que llamó a **FORK**, y esa copia incluye también la pila (no sólo los segmentos de código y datos, recuerden que clona su totalidad).

**Proceso Huerfano**

Cuando un proceso muere antes del hijo que haya creado, este último es “adoptado” y dependiendo el Sistema Operativo puede ser adoptado por el INIT (proceso primario del SO) o asociado al proceso en PID que este antes que el padre muerto.

**Proceso Zombie**

Cuando un proceso termina sin que su padre esté ejecutando un servicio *WAIT*, el proceso queda en estado Zombie. Estos procesos siguen siendo planificables por el SO pero al llegar a EXEC son descartados por tener el estado “Z” de Zombie, son procesos que ocupan lugar y solo pueden ser depurados por un ADMIN o ROOT que los detecte.

**En Resumen**

Los Procesos Hijos son muy efectivos a la hora de acelerar la ejecucion de un proceso pero traen aparejado el uso excesivo de memoria y disco, ya que son un clon de su antecessor.

Aumentan la performace un 50% contra el proceso tradicional.