Procesos Zombies y Huérfanos

C Prácticas, UNIDAD 3 - Sistema de Procesos, Unidad 3 Tema 1

Zombies y Huérfanos

Cuando un proceso muere en linux, no se elimina completamente de la memoria. Su descriptor de proceso permanece en memoria (ocupa muy poco). El estado del proceso que muere pasa a ser EXIT_ZOMBIE y el proceso padre es notificado mediante la señal SIGCHLD que el proceso hijo ha muerto.

Se supone que el proceso padre ejecuta la llamada al sistema wait() para poder leer el estado de terminación del proceso hijo y otra información. Esto permite al proceso padre obtener información del hijo muerto. Después de la wait(), el proceso zombie es eliminado completamente de la memoria.

Esto ocurre muy rápido y por tanto no vemos procesos zombies acumulados en el sistema. Sin embargo, si el proceso padre no está bien programado y no llama a wait(), su hijo zombie se quedará en ese estado usando memoria del sistema que no será eliminada hasta que el padre muera.

Por tanto en el supuesto que el padre continúe su ejecución y los hijos vayan terminando sin que el padre capture la señal SIGCHLD (en cuyo manejador estaría la llamada a wait()) o bien en su código se ejecute wait() en algún momento, a la espera de la terminación de sus hijos, entonces conforme los procesos hijos van terminando van convirtiéndose en zombies.

Cuando el padre muere antes de que los procesos hijos acaben, estos pasan a ser huerfanos (que no zombie), y son heredados por el proceso init que es el proceso padre de todos los procesos. El proceso init tiene el PID = 1. El proceso init ejecuta periódicamente wait()para que todos los hijos huérfanos heredados puedan eliminarse del sistema.

Veamos ejemplos que aparecen en el cuaderno de prácticas

En el ejercicio **c027.c**

```
#include &lt;stdio.h&gt;
#include &lt;unistd.h&gt;
int main(void) {
  printf("Parent PID=%d\n",getpid() );
  if (fork()==0) {
     printf ("PPID=%d\n",getppid() );
     return 0;
     }
return 0;
}
```

El proceso hijo pregunta por el pid de su padre con getppid() y termina. El proceso padre, antes de crear al hijo muestra su pid con getpid(). En la salida, ambos deben de coincidir si el padre no termina antes que su hijo. Si terminase antes que el hijo, el hijo pasaría a ser huérfano y heredado por init, en cuyo caso mostraría el pid 1 (el de su nuevo padre). En función de la carga del sistema, la planificación etc... puede darse el caso, pero normalmente el hijo ejecuta la llamada a getppid() antes de que el padre acabe.

En el ejercicio **c028.c** ...

```
#include &lt;stdio.h&gt;
#include &lt;unistd.h&gt;
int main(void) {
  printf("Parent PID=%d\n",getpid() );
  if (fork()==0) {
     sleep (2);
     printf ("PPID=%d\n",getppid() );
     return 0;
     }
  return 0;
}
```

... forzamos que el hijo duerma unos 2 segundos, tiempo suficiente para que el padre termine y el hijo quede huérfano, siendo por tanto heredado por init. Cuando el hijo despierta muestra el pid de su padre que ahora es 1 y termina quedando zombie. El proceso init será ahora el encargado de ejecutar su wait() para la terminación y eliminación del hijo.

Esto se resuelve parcialmente en el programa **c029.c** ...

```
#include &lt;stdio.h&gt;
#include &lt;unistd.h&gt;
int main(void) {
  printf("Parent PID=%d\n",getpid() );
  if (fork()==0) {
```

```
        printf ("PPID=%d\n",getppid() );
        return 0;
        }
    sleep (2);
    return 0;
}
```

... donde el padre es el que espera los 2 segundos, tiempo suficiente para que el hijo ejecute su llamada a getppid() antes de convertirse en huérfano.

¿Que pasa si el hijo huérfano se quedara en un bucle y no terminara?

Problemas con los procesos zombies

Para ver los procesos zombies que se van quedando en el sistema podemos usar la orden top, ps o bien el Gestor de Tareas. La orden top nos indica en la cabecera cuantos procesos zombies hay. Con ps y el Gestor de Tareas podremos ver el estado de los procesos, que cuando son zombies su estado es Z.

Los procesos zombis no usan recursos del sistema, puesto que cuando mueren el S.O. libera todos los recursos asociados a éstos y elimina la memoria ocupada por el proceso, pero deja el descriptor de proceso y el pid sigue asignado al proceso zombie. Puesto que cada zombie mantiene su pid y el número de procesos distintos es limintado (en un sistema de 32 bits sólo pueden ser 32767 pids distintos). Si los zombies se acumulan muchos y muy rápido podemos poner al sistema en situación de no poder crear más procesos puesto que no quedan pids disponibles para ser asignados a los nuevos procesos.

Sin embargo, tener unos cuantos zombies en un momento dado no es problema, puesto que suponemos que llegarán a ser huérfanos en algún momento e init acabará con ellos. Incluso en el caso en que no, tampoco sería dramático, pero hay que observarlo.

Acabar con los procesos zombies.

No se pueden matar los procesos zombies de la misma forma que puedes matar (con kill) los procesos normales, ya que los procesos zombies ya están muertos. No es necesario eliminar los procesos zombies del sistema a menos que tengas muchos de ellos y continuamente, algún programa mal diseñado esté creando y dejando en bucle o sin terminar, procesos zombies.

En ese caso deberíamos identificar el proceso mal diseñado y corregir el error. Pero tendremos que matar a los procesos zombies, no obstante. Tener en cuenta que el proceso zombie, lo es porque ha muerto y su padre no ha recogido su estado de terminación con wait(), por lo que el hijo zombie, sigue siendo hijo del proceso padre.

Si a un padre le enviamos (como root o como el propietario del proceso) una señal SIGCHLD, este debería lanzar el manejador de la señal donde respondería con una llamada a wait() para liberar al hijo. Pero estamos en el supuesto de que esto no pasa, precisamente por eso se están quedando los hijos zombies. Entonces, no nos queda más remedio que matar al proceso padre que está generando esa gran cantidad zombies.

En el momento que el padre muere, todos los zombies, son a su vez huerfanos, siendo heredados por init, que llamará periódicamente a wait() para garantizar la eliminación de los procesos de memoria.

← Entrada anterior

Entrada siguiente →