Introducción a Sistemas Operativos: Procesos

Clip 19
Francisco J Ballesteros

1. Programas y procesos

A un programa en ejecución se lo denomina *proceso*. El nombre *programa* no se utiliza para referirse a un programa que está ejecutando dado que ambos conceptos (programa y proceso) son diferentes. La diferencia es la misma que hay entre una receta de cocina para hacer galletas y una galleta hecha con la receta. El programa es únicamente una serie de datos (con las instrucciones y datos para ejecutarlo) y no es algo vivo. Por otro lado, un proceso es un programa que está vivo. Tiene una serie de registros, incluyendo un contador de programa, y utiliza una pila para hacer llamadas a procedimiento (y al sistema). Esto significa que un proceso tiene un *flujo de control* que ejecuta una instrucción tras otra, como ya sabes.

La diferencia queda clara si consideras que puedes ejecutar a la vez el mismo programa, varias veces. Por ejemplo, la figura 1 muestra varias ventanas que están ejecutando un shell.

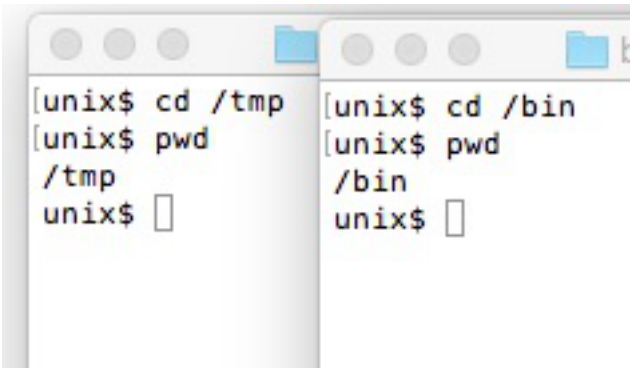

Figura 1: *Varias ventanas ejecutando un shell. Hay solo un programa pero tenemos varios procesos.*

En ambos casos ejecutan el mismo programa: el que está en `/bin/sh`. Pero, aunque hay un sólo programa, tenemos varios procesos. Por ejemplo, si cambiamos de directorio en uno de ellos, el otro sigue inalterado. Cada proceso tiene sus propias variables en la memoria. Y, aunque el programa es el mismo, los valores de las variables serán en general diferentes en cada proceso. Esto es obvio si piensas que cada shell estará haciendo una cosa distinta en un momento dado. No obstante, el programa tiene únicamente un conjunto de variables declaradas.

¿Qué es entonces un proceso? Piensa en los programas que has hecho. Elige uno de ellos. Cuando lo ejecutas, comienza a ejecutar *independientemente* del resto de programas en el ordenador. Al programarlo, ¿has tenido que tener en cuenta otros programas como el shell, el sistema de ventanas, el reloj, el navegador web, ...? ¡Por supuesto que no! Haría falta un cerebro del tamaño de la luna para tener todo eso en cuenta. Dado que no existen esos cerebros, el Sistema Operativo se ocupa de darte como abstracción el *proceso*. Gracias a esa abstracción puedes programar y ejecutar un programa olvidando el resto de programas que tienes en el sistema: Los procesos son programas que ejecutan independientemente del resto de programas que están ejecutando en el sistema.

Cada proceso viene con la ilusión de tener su propio procesador. Naturalmente, esto es mentira y es parte de la abstracción. Cuando escribes un programa piensas que la máquina ejecuta una instrucción tras otra. Y

piensas que toda la máquina es tuya. Bueno, en realidad, que toda la máquina es del programa que ejecuta. La implementación de la abstracción *proceso* es responsable de esta fantasía.

Cuando existen varios procesadores o CPUs, varios programas pueden ejecutarse realmente a la vez, o *en paralelo*. Hoy en día, la mayoría de los ordenadores disponen de múltiples procesadores. Pero, en cualquier caso, seguramente ejecutes más programas que procesadores tienes. ¡Al mismo tiempo! Cuenta el número de ventanas que tienes abiertas y piensa que hay al menos un programa ejecutando por cada ventana. Seguro que no tienes tantos procesadores.

Lo que sucede es que el sistema hace los ajustes necesarios para que cada programa pueda ejecutar durante un tiempo. La figura 2 muestra la memoria del sistema con varios procesos ejecutando.

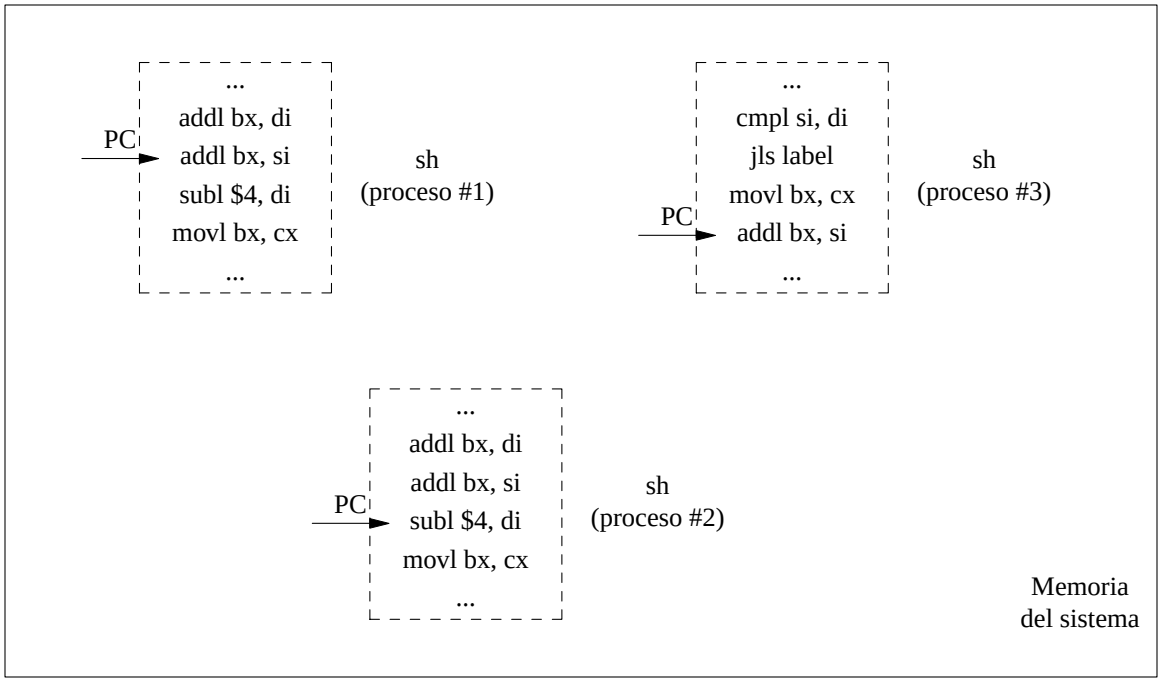

Figura 2: *Ejecución concurrente de varios procesos en el mismo sistema.*

Cada uno tiene su propio juego de registros, incluyendo el contador de programa. Además, cada uno tendrá su propia pila. La figura es tan solo una imagen de la memoria en un momento dado. Durante algún tiempo, el proceso 1 que ejecuta `sh` estará ejecutando en el procesador (sin que el Sistema Operativo intervenga mientras tanto). Después, un temporizador hardware arrancado anteriormente por el sistema interrumpirá la ejecución de dicho programa. En ese momento el sistema puede decidir que el proceso 1 ya ha tenido suficiente tiempo de ejecución por el momento, y en tal caso puede *saltar* para continuar ejecutando (por ejemplo), el proceso 2. Este proceso puede estar también ejecutando `sh`, o cualquier otro programa. Pasado cierto tiempo, otra interrupción de reloj llegará y una vez más se interrumpirá la ejecución del proceso que ejecuta. En este caso, el sistema (donde está el código del programa que atiende la interrupción) puede decidir que ya es hora de volver a ejecutar el proceso 1. De ser así, saltará al punto por el que estaba ejecutando dicho proceso la última vez que ejecutó.

Todo esto sucede detrás del telón. El sistema sabe que hay tan solo un flujo de control por cada procesador, y salta de un programa a otro utilizando dicho flujo. Pero, para los usuarios del sistema, lo que importa es que cada proceso parece estar ejecutando de forma independiente del resto. ¡Como si tuviera un procesador dedicado para el solo!

Como todos los procesos aparentan ejecutar de forma simultánea, decimos que ejecutan *concurrentemente*. O dicho de otro modo, decimos que son procesos concurrentes. En algunos casos, realmente ejecutan a la

vez (cada uno en un procesador). De ser así, decimos que ejecutan *paralelamente*. En la mayoría de los casos compartirán procesador (un tiempo cada uno) y diremos que son *pseudo-paralelos*. Pero, en la práctica, da igual si ejecutan de un modo u otro y simplemente los consideramos concurrentes. A la hora de programar y de utilizar el sistema esto es todo lo que importa.

En este tema vamos a explorar el proceso que obtenemos cuando ejecutamos un programa. Pero antes de hacer esto es importante ver qué hay en un programa y qué hay en un proceso, cosa que haremos a continuación.

2. Planificación y estados de planificación

Supongamos que hay sólo un procesador. De haber varios, lo que vamos a decir se aplica a cada uno de ellos sin más complicación. Cada proceso aparenta ejecutar de forma independiente, con su flujo de control propio, aunque el procesador sólo implementa un flujo de control.

Lo que sucede es que cuando un proceso entra al kernel, ya sea por una interrupción procedente del hardware o por que hace una llamada al sistema, el sistema tiene almacenado el estado del proceso que estaba ejecutando (normalmente en la pila en que se ha salvado el estado en la interrupción o al entrar al sistema). Gracias a esto, el sistema puede decidir que hay que saltar hacia otro proceso y puede *retornar* de la interrupción o llamada al sistema utilizando el estado que se salvó anteriormente para *otro* proceso distinto. Por ejemplo, se puede cambiar la pila por la de otro proceso de tal forma que cuando se retorne sea hacia ese otro proceso. De este modo, a cada proceso se le da un tiempo de procesador y, después, el sistema salta a otro distinto. A esta cantidad de tiempo se la denomina *cuanto*, y puede ser del orden de 100ms (lo que es una enormidad de tiempo para la máquina).

A transferir el control de un proceso a otro, salvando el estado del antiguo y recargando el estado del nuevo, lo denominamos *cambio de contexto*. Esto es obvio si pensamos que cambiamos el *contexto* (registros, pila, etc.) de un proceso a otro. Es de notar que es el kernel el que hace estos cambios de contexto. ¡Tu nunca incluyes saltos en tus programas para que otros programas ejecuten!

La parte del kernel responsable de decidir qué proceso es el siguiente que ejecuta en un cambio de contexto se denomina *planificador*, o *scheduler*. Es fácil si pensamos en que planifica la ejecución de los procesos. A las decisiones del planificador las conocemos como *planificación*, o *scheduling*. ¡Sorprendente! En la mayoría de sistemas el kernel puede sacar del procesador a un proceso incluso si este no hace llamadas al sistema (por ejemplo, mientras ejecuta un bucle). Se suelen utilizar las interrupciones de reloj a tal efecto. En este tipo de planificación, decimos que tenemos un planificador expulsivo (o, en Inglés, *preemptive*). Cuando el planificador no utiliza interrupciones para expulsar procesos decimos que tenemos un planificador *cooperativo*. Pero, en este último caso, un programa que entre en un bucle infinito sin llamar al sistema puede convertir la máquina en un pisapapeles.

Con un único procesador, sólo un proceso puede ejecutar en cada momento. El resto en general pueden estar *listos para ejecutar* (*ready*) pero no estarán ejecutando. Ya conoces dos estados de planificación: *listo* y *ejecutando*. Un proceso ejecutando pasa a estar listo para ejecutar si lo expulsan del procesador cuando su tiempo termina. En ese momento, el kernel ha de elegir a un proceso de entre los que están listos para ejecutar y ha de ponerlo a ejecutar.

Los estados de planificación son tan sólo constantes que define el sistema para guardarlas en los *records* o *structs* con los que implementa los procesos y saber así si puede elegir un proceso para ejecutarlo o no. La figura 3 muestra los estados de planificación de un proceso.

En algunos casos un proceso leerá de teclado o de una conexión de red o cualquier otro dispositivo. Cuando esto ocurre, el proceso habrá de esperar a que el usuario pulse una tecla, a que lleguen datos, o a que suceda alguna otra cosa. El proceso podría esperar entrando en un bucle, pero eso sería un desperdicio de

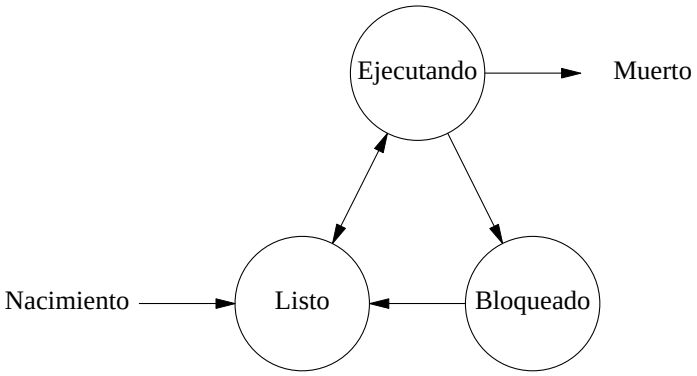

Figura 3: *Estados de planificación de un proceso.*

procesador. En lugar de eso, cuando un proceso llama al sistema y el kernel ve que no puede continuar, se le marca como *bloqueado* (otro estado de planificación) para que el kernel no lo ponga a ejecutar.

Los dispositivos en entrada/salida son tan lentos en comparación con el procesador que es posible ejecutar multitud de cosas mientras un proceso espera a que su entrada/salida termine. A hacer tal cosa se la denomina *multiprogramación*.

Pasado cierto tiempo, un proceso bloqueado volverá a marcarse como listo para ejecutar si el evento que estaba esperando el proceso sucede. Por ejemplo, si el proceso llamó al kernel para leer de teclado y, debido a eso, se bloqueó, en el momento en que escribimos algo en el teclado y el kernel puede dárselo el proceso, dicho proceso pasará a estar listo para ejecutar. La siguiente vez que el planificador ejecute, el proceso es un candidato a ejecutar.

Otra forma de dibujar el estado de los procesos a lo largo del tiempo es utilizar un *diagrama de planificación*. Se trata de dibujar una línea para cada proceso de tal forma que indicamos en qué estado está en cada momento (cambiando la forma o el color de la línea). Naturalmente, si tenemos un procesador, sólo podrá existir un proceso ejecutando en cada momento. Y además, ya sabes que en realidad el salto de un proceso a otro sucede utilizando el mecanismo que hemos descrito antes. Un ejemplo de diagrama de planificación podría ser el que puede verse en la figura 4. En dicha figura, hay dos procesos ejecutando concurrentemente en un sólo procesador. Uno ejecuta `sh` y el otro `ls`.

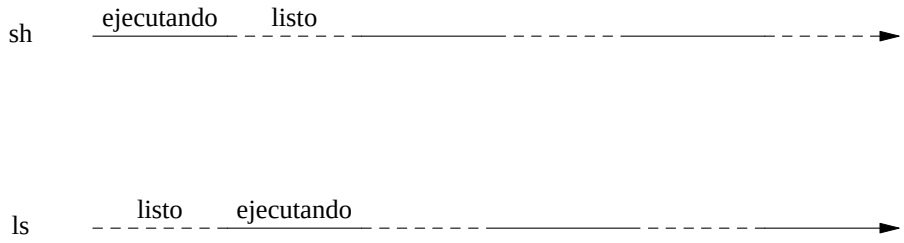

Figura 4: *Diagrama de planificación. El tiempo fluye hacia la derecha en la figura.*