Una aplicación de la conmutación de tareas: aprovechamiento de herramientas comerciales en programación

Carlos Barahona

Aunque quizás sin percatarse, la mayoría de usuarios de ordenadores han empleado alguna vez la técnica de conmutación de tareas: se trata, sencillamente, de suspender temporalmente la ejecución de un programa para trabajar con otro, de forma que, en cualquier momento, podemos volver a operar con el primero en el mismo punto en que lo habíamos dejado. Esto ocurre, por ejemplo, cuando, trabajando con un procesador de textos en Windows, pulsamos las teclas *Alt+Tab* para jugar una partida al solitario.

Otra aplicación de la conmutación de tareas es la que se expone en este artículo: el aprovechamiento de las rutinas o servicios de *software* comercial desde un programa creado por nosotros. De esta forma nos podemos ahorrar mucho tiempo y trabajo en programación, sacando partido a otros programas ya creados. En este caso, además de la conmutación, tendremos que buscar formas de transferir órdenes y datos entre el *software* comercial y nuestro programa.

CARLOS BARAHONA *es ingeniero Técnico en Equipos Electrónicos por la EUP de Vilanova i la Geltrú. Actualmente continua sus estudios en la ETSETB.*

Esta técnica es distinta según la queramos aplicar bajo entorno MS-DOS o bajo entorno Windows.

INTRODUCCIÓN A LA CONMUTACIÓN DE TAREAS.

En nuestro caso, por tarea entenderemos un programa cualquiera cargado en memoria (no necesariamente memoria RAM, por ejemplo, Windows utiliza parte de disco duro como si fuera memoria RAM). El estado de una tarea será entonces, el conjunto de valores de los registros del microprocesador en un momento determinado. Notar que entre estos registros se encuentran, entre otros, el puntero de dirección de ejecución y el puntero de pila.

Supongamos ahora, que, en un cierto punto de ejecución de una tarea tomamos su estado y, de alguna forma, suspendemos su ejecución para cederle el control a otra. Si al cabo de cierto tiempo, actualizamos los registros del microprocesador con los valores capturados anteriormente, volveremos a la primera tarea justo en el punto donde se había quedado: es como si el lapso de tiempo que ha estado inactiva no hubiera ocurrido. En verdad, esto será cierto si tomamos la precaución de no modificar los segmentos de datos y de código, y, sobretodo, la pila de esta tarea.

La implementación de este sistema, para el caso de dos tareas, se muestra en la figura 1. En este esquema aparece una zona de memoria común a ambas tareas. Este bloque de memoria será empleado para transferir datos, entre ellos, el estado en que queda la tarea que va a ser suspendida, para que la tarea entrante pueda devolverle el control.

La programación de esta técnica es bastante sencilla en código máquina. Aún así, los lenguajes de alto nivel también incluyen instrucciones que realizan las funciones de guardar y actualizar los registros del microprocesador (por ejemplo, en Borland C, disponemos de las funciones *longjmp* y *setjmp*).

Si utilizamos esta técnica bajo entorno MS-DOS, hemos de tomar ciertas precauciones. Primero, nunca se debe intentar volver a una tarea si el sistema operativo la considera finalizada. Esto ocurre cuando se ejecutan instrucciones como *exit* o *abort*. En estos casos, aunque es posible que no se haya liberado la memoria de la tarea, es decir, su código y datos siguen presentes, sus ficheros de entrada/salida (incluidos los que controlan el vídeo) se cierran, provocando, generalmente, el bloqueo del sistema.

Lo que sí debemos hacer al conmutar de tarea, es informar al sistema operativo de lo ocurrido. Esto se hace actualizando el valor de dos variables de éste: la dirección del PSP y la del DTA, para lo que disponemos de interrupciones como GET_PSP o SET_PSP.

Si trabajamos bajo entorno Windows, se simplifica bastante la programación, pues será éste el que se encargue de todas las operaciones comentadas anteriormente. Para realizar la conmutación de tareas debemos seguir dos pasos: primero, informar a Windows de nuestras intenciones. Para ello, es necesario enviar y contestar correctamente a una serie de mensajes especiales, según un cierto protocolo. Por suerte, la función del Windows API *SetActiveWindow* se encarga de todo el proceso.

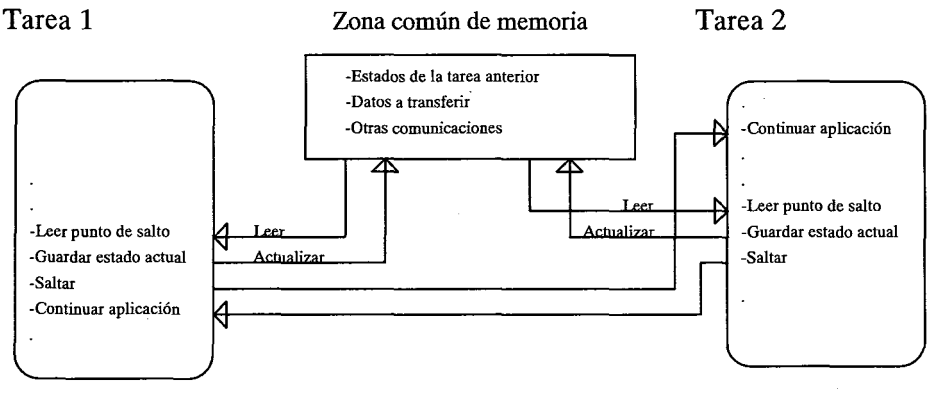

Figura 1.-Esquema general de conmutación entre dos tareas.

Segundo, para que realmente se produzca la conmutación, hemos de indicar a Windows que la tarea actualmente activa no tiene nada más que hacer por el momento. Para lograrlo, lo más seguro es que entre en un bucle de mensajes, igual al que utilizamos en la función principal.

IMPLEMENTACIÓN CON *SOFTWARE* COMERCIAL.

Hasta ahora hemos considerado que las dos tareas entre las que conmutamos eran capaces de, en un momento determinado, acceder a los registros del microprocesador o enviar mensajes. Esto no suele ser cierto cuando se trata de *software* comercial, que era precisamente nuestro objetivo. Para salvar esta dificultad tendremos que realizar estas operaciones desde un programa externo (que llamaremos controlador).

Esto supone una restricción en cuanto al tipo de *software* comercial que podemos emplear: es necesario que sea capaz de ejecutar un programa externo. No se trata de una restricción desesperanzadora, pues la mayoría de los programas comerciales incorporan esta propiedad.

TRANSFERENCIA DE ÓRDENES Y DATOS.

Nos queda por resolver como indicar al *software* comercial que instrucciones debe ejecutar y con que datos. En cuanto a la transferencia de los datos, lo más sencillo es mediante ficheros de disco: todos los programas tienen comandos para almacenar los resultados obtenidos en algún tipo de fichero. Nuestro trabajo será averiguar el formato de estos ficheros.

La transferencia de órdenes es algo más complicada si trabajamos con MS-DOS. En realidad, no hay ningún sistema general, dependerá del *software* que estemos empleando. Si se trata de un entorno de programación (como un paquete matemático o un intérprete Basic) , por ejemplo, es casi seguro que dispondrá de un tipo de fichero donde almacene los programas de usuario. Podemos entonces hacer que entre en un bucle que continuamente ejecute el controlador y, seguidamente, uno de estos ficheros con un programa de usuario. Cada vez que se llame al controlador, se ejecutará nuestro programa que deberá crear o modificar el fichero con las órdenes precisas.

En el caso de trabajar con Windows, sí que disponemos de un sistema, más sencillo, y apto para casi cualquier tipo de *software*: basta con enviarle mensajes de teclado. El paquete comercial interpreta estos mensajes como si provinieran de un usuario que estuviera tecleando. De esta forma, cualquier operación que un usuario pueda realizar con un programa, la podremos programar nosotros con tan sólo enviar la secuencia de mensajes adecuada.

UN CASO PRÁCTICO.

La técnica mostrada ha sido empleada por el Departamento de Ingeniería Electrónica en la escuela universitaria de Vilanova i la Geltrú, dentro de una línea de investigación dedicada al estudio y potenciación de herramientas destinadas a la simulación de sistemas.

El objetivo es la creación de un entorno donde poder integrar diversos algoritmos de simulación, desarrollados por diversas fuentes (proyectistas, profesores, etc.), aprovechando herramientas ya existentes (paquetes comerciales).

Para lograr este objetivo se ha pensado en un esquema. En la base tenemos el entorno de trabajo, que, en este caso, será un paquete matemático comercial (Matlab). La misión de este paquete será dar soporte a la programación de las diversas rutinas, aportando los algoritmos matemáticos necesarios en simulación.

Entre las distintas herramientas y el entorno de trabajo se encuentran las librerías de enlace. Éstas son un conjunto de funciones encargadas de facilitar el acceso al paquete matemático desde programas externos. Para ello se emplean técnicas como la expuesta aquí. De hecho, el objetivo de estas librerías es hacer que los distintos comandos matemáticos disponibles en el entorno de trabajo aparezcan como una ampliación de las funciones del lenguaje de programación empleado en el desarrollo de las herramientas.

Por último, el entorno de usuario será el encargado de dar coherencia a todo el paquete, facilitando el acceso del usuario a cualquiera de las herramientas diseñadas.