Concurrencia y distribución

Hoy en día, cualquier usuario espera poder hacer varias cosas a la vez y no verse forzado a ejecutar los programas secuencialmente. Aun cuando un PC suele disponer únicamente de un microprocesador, los sistemas operativos multitarea como Windows se encargan de que varios programas se puedan ejecutar concurrentemente incluso cuando sólo se dispone de una única CPU.

Dos tareas se dice que son concurrentes si transcurren durante el mismo intervalo de tiempo. Se dice que dos tareas se ejecutan concurrentemente cuando se ejecutan "a la vez". Obviamente, la simultaneidad de la ejecución de las distintas tareas es totalmente ficticia. En realidad, el sistema operativo pasa el control de la CPU de una tarea a otra cada pocos milisegundos, algo conocido como cambio de contexto. Al realizar los cambios de contexto en intervalos de tiempo muy cortos para nosotros, el usuario tiene la percepción de que las distintas tareas se ejecutan en paralelo (algo que, obviamente, sólo sucede si disponemos de un multiprocesador o de un multicomputador).

Cuando decidimos descomponemos un programa en varias tareas potencialmente paralelas, estas tareas las podemos implementar en el ordenador como procesos o como hebras:

- Un **proceso** es un programa en ejecución con un estado asociado. Las distintas aplicaciones que se pueden ejecutar en un sistema operativo multitarea son procesos independientes. Con independientes queremos decir que cada una de ellas ocupa un espacio de memoria independiente (para no interferir con la ejecución de otros procesos). En realidad, una aplicación puede implementarse

como un conjunto de procesos independientes en el sentido anterior pero que colaboren entre sí para lograr sus objetivos, para lo que se pueden emplear distintos **mecanismos de comunicación entre procesos** (entre los que se encuentran los sockets TCP/IP, .NET Remoting y los Servicios Web, entre otros muchos).

- Los sistemas operativos actuales permiten un nivel adicional de paralelismo dentro de un proceso: en un proceso pueden existir varias **hebras** de control independientes. Cada hebra es una vía simultánea de ejecución dentro del espacio de memoria del proceso. En este caso, la comunicación entre las distintas hebras se puede realizar a través del espacio de memoria que comparten, aunque habrá que utilizar **mecanismos de sincronización** para controlar el acceso a este recurso compartido entre las distintas hebras de un proceso.

Teniendo en cuenta lo anterior, se denomina aplicación concurrente a una aplicación que se descompone en un conjunto de procesos y/o hebras. Del mismo modo, una aplicación multihebra está constituida por distintas hebras que comparten el espacio de memoria de un proceso. Al existir varios procesos o hebras, el estado de la aplicación en cada momento vendrá determinado por el estado de cada uno de los procesos o hebras que la componen, lo que puede dificultar la realización de tareas como la depuración de programas si no se toman las medidas adecuadas.

En cada momento, una hebra o proceso tiene asociado un contexto que incluye el estado interno de los registros del procesador, la lista de interrupciones y señales que admite, y una pila de ejecución independiente. Todos estos datos son los que el sistema operativo ha de guardar cada vez que se realiza un cambio de contexto. En el caso de un proceso, su contexto incluye, además, el conjunto de recursos del sistema que esté utilizando, entre los que se encuentran las páginas de memoria RAM que ocupa y los ficheros que tenga abiertos.

Los cambios de contexto son costosos especialmente en el caso de los procesos (al tener que cambiar el espacio de direcciones en el que se trabaja). Por este motivo, y meramente por cuestiones de eficiencia, en muchas ocasiones se prefiere la utilización de hebras. Además, la comunicación entre procesos que se ejecutan en distintos espacios de memoria es más costosa que cuando se realiza a través de la memoria que comparten las distintas hebras de un proceso, como resulta evidente. Como contrapartida, un fallo en una hebra puede ocasionar la caída completa de la aplicación mientras que, si se emplean procesos independientes, los fallos se pueden contener en el interior de un proceso, con lo que la tolerancia a fallos de la aplicación será mayor.

Independientemente de si utilizamos procesos o hebras, el desarrollo de aplicaciones concurrentes involucra el uso de técnicas específicas y la superación de dificultades que no se presentan en la implementación de programas secuenciales. A la hora de crear aplicaciones concurrentes, distribuidas o paralelas, deberemos tener en mente ciertas consideraciones:

- El diseño de aplicaciones concurrentes es más complejo que el de aplicaciones secuenciales, ya que hemos de descomponer el programa en un conjunto de tareas más o menos independientes posibles con el fin de aprovechar el paralelismo que

pueda existir. Si no existe ese paralelismo potencial, no tiene sentido que intentemos descomponer nuestra aplicación en tareas independientes.

- La implementación de aplicaciones concurrentes es también más compleja que la de aplicaciones secuenciales convencionales porque hemos de garantizar la coordinación de las distintas hebras o procesos con los mecanismos de comunicación adecuados, además de velar por la integridad de los datos con los que éstas trabajan simultáneamente (para lo cual hemos de sincronizar el acceso a los mismos).

- La depuración de las aplicaciones concurrentes es extremadamente difícil, dado que la ejecución de los distintos procesos/hebras se realiza de forma independiente y las operaciones que realizan se pueden entrelazar de cualquier forma en función de cómo les asigne la CPU el ordenador (algo que no podemos prever por completo).

- En tiempo de ejecución, además, cada hebra o proceso supone una carga adicional para el sistema, por lo hay que tener en cuenta la eficiencia de la implementación resultante. Deberemos ser cuidadosos para asegurar que se aprovecha el paralelismo para mejorar el rendimiento de la aplicación. Este rendimiento puede medirse en función del tiempo de respuesta del sistema o de la cantidad de trabajo que realiza por unidad de tiempo [*throughput*].

Desafortunadamente, los entornos de programación actuales no siempre ofrecen demasiada ayuda para el desarrollo de aplicaciones concurrentes. Por suerte, existen técnicas que podemos aprender a manejar para facilitar la creación de aplicaciones de este tipo, como describiremos a continuación.

La mayor dificultad con la que nos encontramos a la hora de trabajar con aplicaciones concurrentes es el hecho de que la ejecución asíncrona de las distintas hebras o procesos da lugar a un conjunto de interacciones difícil de analizar. Estas interacciones dan lugar a algo se suele denominar *comportamiento emergente* en el estudio de sistemas complejos). Para analizar este comportamiento emergente, se pueden utilizar técnicas de modelado como los diagramas de secuencias y los diagramas de estado incluidos en UML, el lenguaje unificado de modelado. Estas herramientas tiene su base en métodos formales de análisis como las máquinas de estados o las redes de Petri, que nos permiten comprobar que nuestras aplicaciones posean las propiedades deseables que de ellas se espera:

- **Corrección**: Si bien la corrección de un programa es algo que no se puede demostrar matemáticamente, por ejemplo, se puede analizar un programa para detectar posibles inconsistencias en el acceso sincronizado a recursos compartidos.

- **Vivacidad**: Esta propiedad hace referencia a que el programa debe avanzar hasta proporcionar una respuesta. En el caso de los programas secuenciales, la existencia de un bucle infinito es la única causa posible de falta de vivacidad. Cuando trabajamos con programas que se ejecutan concurrentemente, también

pueden aparecer bloqueos [*deadlocks*] que se ocasionan cuando cada proceso o hebra se queda esperando a que otro de los procesos o hebras también bloqueados libere un recurso compartido al que quiere acceder.

Las dos propiedades mencionadas, corrección y vivacidad, deben estar a la cabeza de nuestra lista de prioridades. Igual que sucede en el desarrollo de aplicaciones secuenciales, es más fácil comenzar por una implementación correcta e intentar mejorar su eficiencia que crear una implementación extremadamente eficiente en la que luego debamos buscar los errores. Además, la búsqueda de errores en una aplicación concurrente es muy difícil. Ni la realización de pruebas de unidad ni el uso de depuradores convencionales sirve de mucho en estos casos, porque los errores pueden provenir de la forma en la que se entrelaza la ejecución de las distintas hebras y procesos, dando lugar a fallos no reproducibles con las herramientas habituales. De hecho, es relativamente habitual en aplicaciones multihebra que se encuentren errores que se producen sólo al usar el depurador o que se detectan al ejecutar la aplicación pero desaparecen al depurar. Esos errores suelen depender de la temporización de operaciones realizadas por la aplicación y no son fácilmente reproducibles.

Obviamente, la forma de eliminar este tipo de errores es construir programas secuenciales, que son completamente seguros en este sentido, si bien podemos emplear algunas técnicas que resultan de utilidad. Por ejemplo, podemos incluir aserciones en nuestro código que verifiquen el cumplimiento de condiciones que deben mantenerse siempre. De esta forma, en cuanto se detecte una desviación respecto al comportamiento deseado nos daremos cuenta de forma inmediata (algo que también resulta útil en el desarrollo de programas secuenciales).

Por otro lado, para garantizar la reproducibilidad de los errores que se produzcan, podemos trazar secuencias de eventos, las interacciones entre hebras o procesos paralelos. Existen varias formas de realizar esas trazas, ya sea introduciendo código adicional de forma manual o interponiendo intermediarios [*proxies*] entre los procesos comunicantes. Una vez detectado el error, se puede usar la traza de eventos para reproducir las circunstancias en las que se produjo inicialmente el error (las "condiciones de carrera").

Aparte de los problemas mencionados, que suelen estar relacionados con el uso de recursos compartidos por parte de hebras o procesos no del todo independientes, el desarrollo de aplicaciones concurrentes también involucra el uso de mecanismos de comunicación. Estos mecanismos, conocidos genéricamente como mecanismos de comunicación entre procesos (IPC) permiten que los distintos procesos que conforman una aplicación "hablen entre sí". Esos procesos, además, pueden ejecutarse en máquinas diferentes. En este caso, la comunicación involucra el uso de redes de ordenadores, por lo que también deberemos analizar el impacto que los protocolos de comunicación utilizados tienen en la comunicación entre los distintos procesos.

Vistos con cierto nivel de detalle, los distintos tipos de sistemas concurrentes que existen pueden parecer muy diferentes entre sí. No obstante, al final, y utilicemos la plataforma que utilicemos, dispondremos de una serie de mecanismos que nos permitan realizar las siguientes cinco operaciones:

- Crear una hebra o proceso.

- Garantizar la exclusión mutua en el acceso a un recurso compartido.
- Detener la ejecución de una hebra/proceso en espera de que se produzca algún tipo de evento.
- Reanudar la ejecución de la hebra/proceso al producirse el evento que estaba esperando.
- Destruir una hebra o proceso (sólo como medida de emergencia).

Pese a que el desarrollo de aplicaciones concurrentes tenga fama de resultar "exótico y difícil", sólo se necesitan algunas herramientas básicas y algo de disciplina para evitar los errores más comunes. En los siguientes capítulos veremos cómo se pueden usar esas herramientas básicas en la práctica para resolver problemas reales en la plataforma .NET.

Desde un punto de vista teórico, se entiende por programación concurrente el conjunto de técnicas y notaciones que sirven para expresar el paralelismo potencial en los programas, así como resolver problemas de comunicación y sincronización.

Cuando se trabaja en entornos distribuidos o con máquinas con múltiples procesadores, se suele hablar de programación distribuida o paralela, respectivamente, si bien los principios y técnicas utilizados, así como los problemas con los que nos encontramos en el desarrollo de aplicaciones de este tipo seguirán siendo los mismos con los que hemos de enfrentarnos al crear aplicaciones que se ejecuten concurrentemente en una única máquina.