Concurrencia

Procesos y hebras

- Concurrencia
- Programación concurrente
- ¿Por qué usar hebras y procesos?

Ejecución de procesos

Ejecución de hebras

- Hebras vs. Procesos
- Creación y ejecución de hebras
- La prioridad de las hebras
- Finalización de la ejecución de una hebra

Uso de recursos compartidos

- Mecanismos de exclusión mutua
- `synchronized`

Hebras e interfaces de usuario

- `SwingUtilities.invokeLater()`
- `javax.swing.Timer`

Más información…

Procesos y hebras

Hoy en día, cualquier usuario espera poder hacer varias cosas a la vez y no verse forzado a ejecutar los programas secuencialmente.

Los sistemas operativos multitarea, como Windows o UNIX, se encargan de que varios programas se puedan ejecutar a la vez (*concurrentemente*) incluso cuando sólo se dispone de una única CPU.

Concurrencia

Dos **tareas** se dice que son **concurrentes** si transcurren durante el mismo intervalo de tiempo.

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.

NOTA: En un PC, 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, 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.
  - Cada proceso ocupa un espacio de memoria independiente (para no interferir con la ejecución de otros procesos).
  - Una aplicación puede implementarse como un conjunto de procesos que colaboren entre sí para lograr sus objetivos, para lo que se pueden emplear distintos ***mecanismos de comunicación entre procesos***.

- Los sistemas operativos actuales permiten un nivel adicional de paralelismo dentro de un proceso: En un proceso pueden existir varias **HEBRAS** de control independientes [*threads*].
  - Cada hebra es una vía simultánea de ejecución dentro del espacio de memoria del proceso.
  - 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 por todas las hebras de un proceso.

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.

Programación concurrente

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 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.

- La **depuración** de las aplicaciones concurrentes es extremadamente difícil, dado que la ejecución de los distintos procesos/hebras se entrelaza conforme el sistema operativo les va asignando la CPU (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*].

¿Por qué usar hebras y procesos?

El no determinismo introducido por el entrelazado de las operaciones de las distintas hebras o procesos de una aplicación concurrente provoca la aparición de errores difíciles de detectar y más aún de corregir: la vida del programador resultaría mucho más sencilla si no hiciese falta la concurrencia.

Sin embargo, existen razonas por las cuales es aconsejable utilizar procesos y hebras:

1. De cara al usuario

Hebras y procesos permiten la creación de interfaces que respondan mejor a las órdenes del usuario.

Cuando una aplicación tiene que realizar alguna tarea larga, su interfaz debería seguir respondiendo…

- La ventana de la aplicación debería refrescarse y no quedarse en blanco (como pasa demasiado a menudo).
- Los botones existentes para cancelar una operación deberían cancelar la operación de un modo inmediato (y no al cabo de un rato, cuando la operación ya ha terminado de todos modos).

Si nuestra aplicación ha de atender las peticiones de distintos usuarios (como sucede, por ejemplo, en un servidor web), el uso de hebras o procesos permite que varios usuarios accedan simultáneamente a la aplicación sin tener que esperar turno.

2. Aprovechamiento de los recursos del sistema

Cualquier operación que pueda bloquear nuestra aplicación durante un período de tiempo apreciable es recomendable que se realice de forma independiente.

- La CPU es el dispositivo más rápido del ordenador: Desde su punto de vista, todos los demás dispositivos del ordenador son lentos, desde una impresora hasta un disco duro con Ultra-DMA.

- En un entorno distribuido, las operaciones de E/S son aún más lentas, ya que el tiempo necesario para acceder a un recurso disponible en otra máquina a través de una red depende, entre otros factores, de la carga de la máquina a la que accedemos y del ancho de banda del que dispongamos.

Paralelismo real

Su aprovechamiento requiere el uso de hebras y procesos…

- Un sistema multiprocesador dispone de varias CPUs.

- Un cluster de ordenadores está compuesto por un conjunto de ordenadores conectados entre sí, a los que se accede como si se tratase de un único ordenador.

- Algunas versiones del Pentium 4 de Intel incorporan *Simultaneous MultiThreading* (SMT), que Intel denomina comercialmente *Hyper-Threading*, con lo que un único microprocesador puede funcionar como si tuviésemos un multiprocesador.

3. Modularización: Paralelismo implícito

A veces, los motivos que nos llevan a utilizar hebras o procesos no tienen una justificación física, sino que un programa puede diseñarse con más comodidad si lo descomponemos en un conjunto de tareas independientes.

Un ejemplo mundano…

> El problema de perder peso puede verse como una combinación de dos actividades: hacer ejercicio y mantener una dieta equilibrada. Ambas deben realizarse durante el mismo período de tiempo, aunque no necesariamente a la vez.

En situaciones de este tipo, la concurrencia de varias actividades es la solución adecuada para un problema: la solución natural al problema implica concurrencia, mientras que una solución secuencial sería extremadamente difícil.

> Ejemplos: Simulaciones, videojuegos…

IMPORTANTE

El objetivo principal del uso de paralelismo es
mejorar el rendimiento del sistema.

El uso de paralelismo (hebras o procesos) permite mejorar el tiempo de respuesta de una aplicación
o la carga de trabajo que puede soportar el sistema.

No obstante, un número excesivo de hebras o procesos puede llegar a degradar el rendimiento del sistema (debido al tiempo de CPU requerido por los cambios de contexto): hay que analizar hasta qué punto compensa utilizar hebras y procesos.