Grado en Ingeniería Informática - 3er Curso

Concurrencia y distribución

Martín Pérez Pérez (martiperez@uvigo.es)

Noelia García Hervella (nghervella@esei.uvigo.es)

Guillermo Blanco González (guillermo.blanco.gonzalez@uvigo.gal)

Repaso - Monitor

En Java, el término "monitor" se refiere a un mecanismo utilizado en programación concurrente para garantizar la exclusión mutua entre múltiples hilos que acceden a recursos compartidos.

En esencia, un monitor proporciona un mecanismo para sincronizar el acceso concurrente a secciones críticas de código, asegurando que solo un hilo pueda ejecutar estas secciones a la vez.

Repaso - Synchronized

Cada objeto derivado de la clase Object (esto es, practicamente todos) tienen asociado un elemento de sincronización o lock intrínseco, que coordina el acceso de múltiples hilos al objeto por lo que afecta a la ejecución de los métodos o bloques definidos como synchronized en el objeto:

- Cuando un objeto ejecuta un método synchronized, toma el control del monitor, y cuando termina de ejecutarlo lo libera.
- Cuando un hilo tiene el control del monitor, ningún otro hilo puede ejecutar ningún otro método synchronized del mismo objeto.
- El hilo que ejecuta código protegido por synchronized de un objeto cuyo monitor se encuentra tomado, se suspende hasta que el objeto es liberado y se le concede el acceso.

Repaso - Synchronized

- Cuando un hilo tiene el control sobre el monitor de un objeto, puede acceder a los demás métodos synchronized de ese mismo objeto.
- Cada instancia de un objeto tiene su propio monitor asociado, por lo que, los monitores son específicos para cada objeto individual y no se comparten entre las instancias. Cada vez que un hilo necesita acceder a un objeto, adquiere el monitor de esa instancia específica del objeto.
- Los métodos de clase (static) también pueden ser synchronized. Por cada clase hay un monitor y es relativo a todos los métodos synchronized de la clase. Este monitor no afecta a los accesos sobre métodos synchronized de los objetos que son instancia de la clase.

Repaso - Synchronized

- Ventaja: maneja automáticamente el bloqueo y la liberación del monitor asociado con el objeto sincronizado.
- Desventaja: menor flexibilidad que otros métodos, ya que synchronized se aplica a un bloque de código completo.

Sincronización por condición

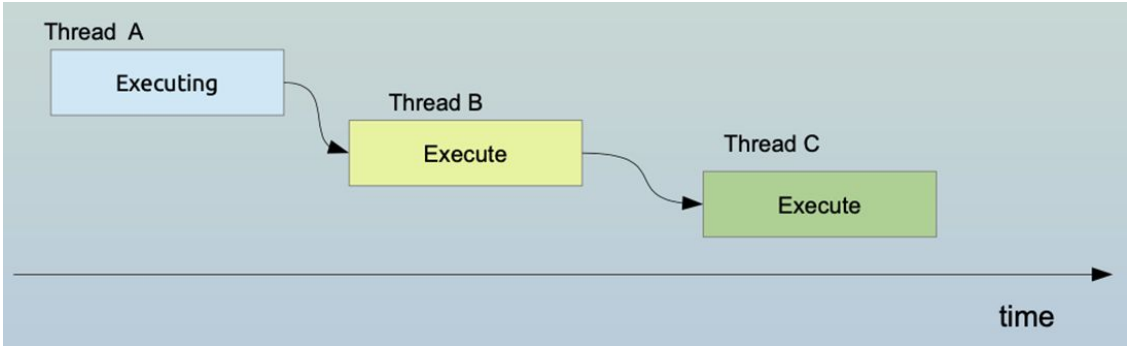
La sincronización por condición en Java se refiere al mecanismo mediante el cual los hilos pueden **esperar** y **notificarse** entre sí en función de ciertas condiciones. Se basa en monitores, que permiten que un **hilo espere hasta que se cumpla una condición** antes de continuar su ejecución.

Monitor - condición de espera y notificación

Los monitores a menudo incluyen mecanismos para que los hilos esperen hasta que se cumplan ciertas condiciones antes de continuar la ejecución. Esto se logra a través de métodos como **wait()** y **notify()** o **notifyAll()**. Los hilos pueden esperar en una condición específica dentro del monitor y ser notificados por otro hilo cuando la condición cambia y es seguro continuar.

Señalización de hilos

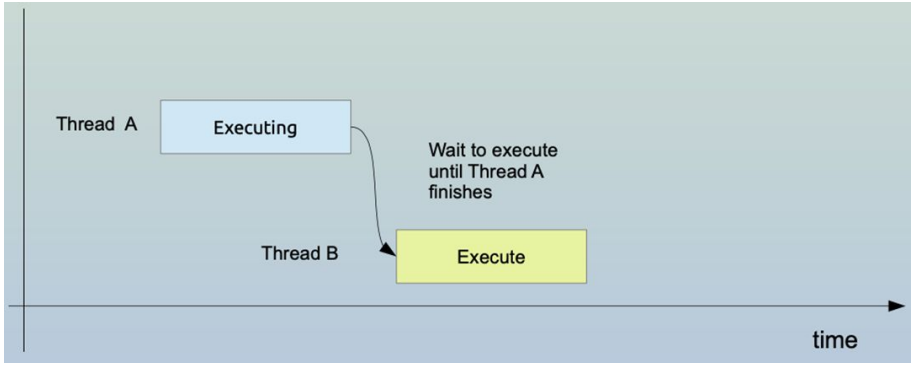
- Podemos controlar secciones críticas.
- Pero, ¿cómo podemos controlar el orden de ejecución de los hilos?
- Se quiere comunicar el estado de un hilo a otro para que se ejecute el siguiente hilo.



Señalización de hilos

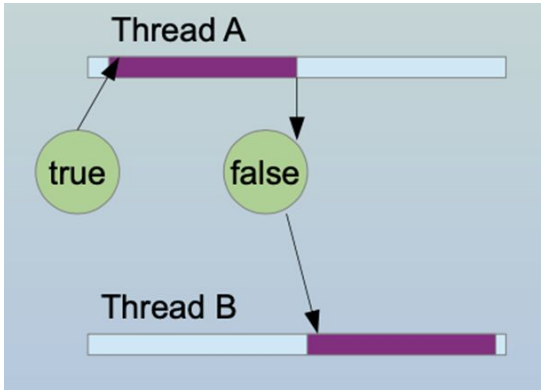
El Thread Signaling es una técnica utilizada para coordinar la ejecución de múltiples hilos de un programa. Permite que **un hilo notifique a otro(s)** cuando ocurre un evento o cuando ciertos datos están listos, evitando que los hilos tengan que estar verificando constantemente una condición (lo que sería ineficiente).

- Por ejemplo, un hilo B podría esperar una señal del hilo A que indique que los datos están listos para ser procesados.



Señalización de hilos

- Una manera sencilla que se puede utilizar para que los hilos se envíen señales entre sí es estableciendo valores de la señal en alguna variable compartida o que forme parte de un objeto compartido.
- El hilo A puede cambiar el valor de una variable booleana desde el interior de un bloque sincronizado, y el subproceso B puede leer esa variable, para determinar sus propias acciones a tomar.

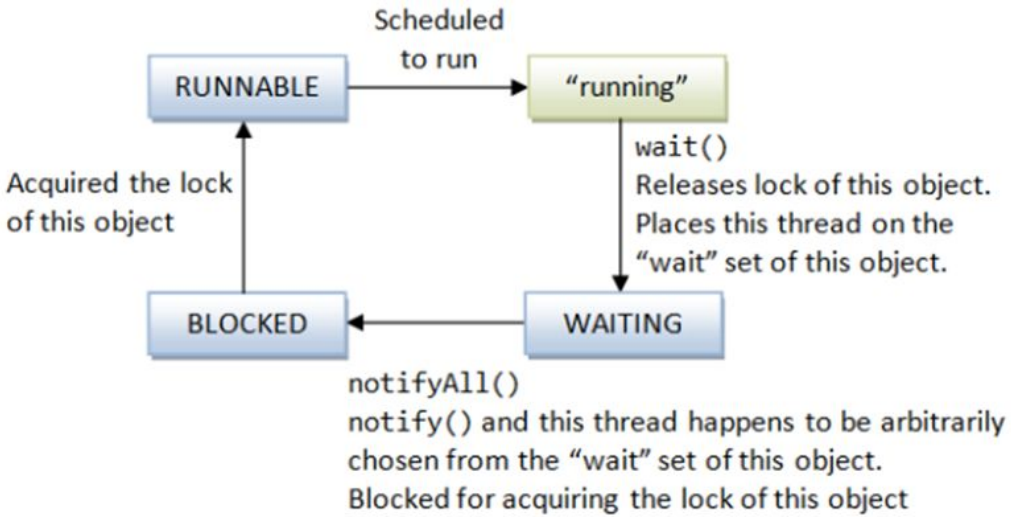


Señalización de hilos

Esto quiere decir que:

- Si usamos una variable booleana como una "señal" para indicar cuándo un hilo debe actuar.
- El hilo A cambia el valor de la variable **dentro de un bloque sincronizado**.
- El hilo B lee la variable y decide qué hacer basándose en su valor.
- Se necesita sincronización para asegurar que el valor actualizado de la variable sea visible para todos los hilos (evitar condiciones de carrera y problemas de caché de CPU).

Ciclo de vida de un hilo en Java



wait(), notify() y notifyAll()

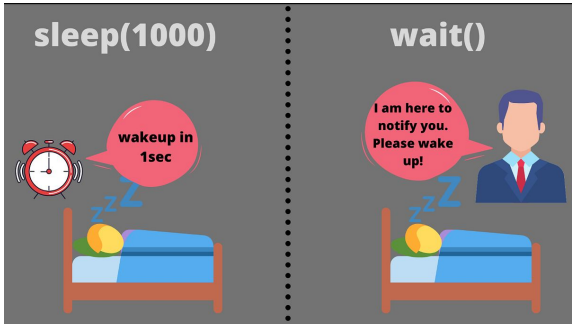
Java proporciona mecanismos que permiten que varios hilos se puedan coordinar y sincronizar. **Wait**(), **notify**() y **notifyAll**() son herramientas potentes que permiten realizar estas tareas.

Estos métodos son fundamentales para la implementación de patrones de concurrencia como el **productor-consumidor**, el bloqueo de recursos compartidos y la sincronización de actividades entre múltiples hilos. Permiten a los hilos cooperar y coordinar sus acciones de manera que se eviten condiciones de carrera y se garantice la consistencia de los datos.

wait()

wait() actúa como un elemento fundamental para la sincronización eficaz de subprocesos, permitiendo al hilo actual liberar el bloqueo y esperar a que otro hilo lo notifique.

Su uso radica en hacer que un hilo aguarde hasta que otro hilo lleve a cabo una acción determinada, como la alteración de un estado compartido o la disponibilidad de un recurso.



wait()

Al llamar a wait(), el hilo se suspende y libera el bloqueo del objeto asociado, permitiendo que otros hilos accedan a recursos compartidos de manera segura. Este método es vital para evitar interbloqueos y garantizar una sincronización eficiente

- **Evitar interbloqueos**: se produce cuando varios hilos están bloqueados indefinidamente, esperando que se liberen recursos que otros hilos poseen, y al mismo tiempo, no están liberando los recursos que ellos mismos poseen. Al liberar al objeto se permite el acceso de otros hilos a los recursos compartidos.

wait()

- **Sincronización eficiente**: en algunas ocasiones los hilos esperan de manera activa a que ocurra un evento específico consumiendo recursos del procesador de manera innecesaria. Usando wait() se induce al hilo a que entre en un estado de espera para que espere de manera eficiente que un evento específico tenga lugar, es decir, que el hilo no se reactiva, sino que espera a que otro hilo le notifique que la condición ya se cumple y puede volver a trabajar.

wait() - ejemplo

Digamos que hay dos hilos “Emisor” y “Receptor” ambos comparten un objeto común “mensaje”.

1. El “Receptor” lee el mensaje cuando no está vacío, de lo contrario tiene que esperar a que el “Escritor” redacte el archivo “mensaje”.
2. Por otro lado, “Escritor” escribe un mensaje cuando está vacío si no espera a que “Receptor” lea el mensaje y el mensaje esté vacío.

wait()

Se utiliza para hacer que un subproceso abandone voluntariamente su bloqueo sobre un objeto, permitiendo que otro subproceso ejecute código dentro de un bloque sincronizado.

- El hilo que llama a wait() entrará en un estado de espera hasta que otro subproceso llame a notify() o notifyAll() en el mismo objeto, lo que le permitirá reanudar la ejecución.
- Este método debe utilizarse siempre dentro de un bloque o método synchronized. A mayores suele implementarse dentro de un ciclo indefinido que verifica una condición.

wait()

- Cuando un hilo llama a wait() no entra solamente en un estado de espera, sino que se coloca en una cola de preejecución.
- Permitir al hilo actual liberar el bloqueo del objeto asociado, hace alusión a la liberación del monitor de dicho objeto.
- La suspensión del hilo y la liberación del monitor son en sí una acción atómica, es decir, nada puede ocurrir entre estas. Así mismo, cuando el hilo se despierta como consecuencia de una notificación, la activación de este y la toma del monitor del objeto son también atómicos.

wait()

- **wait()**: hace que el hilo actual espere hasta que otro hilo invoque el notify()método o el notifyAll()método para este objeto.
- **wait(long timeout)**: hace que el subproceso actual espere hasta que otro subproceso invoque el notify() método o el notifyAll()método para este objeto, o hasta que haya transcurrido una cantidad de tiempo específica.

Wait y Sleep no son lo mismo

Característica	wait()	sleep()
Ubicación	Método de la clase Object	Método de la clase Thread
Necesita synchronized?	✅ Sí, debe llamarse dentro de un bloque sincronizado (synchronized)	❌ No requiere sincronización
Libera el monitor/bloqueo?	✅ Sí, libera el bloqueo del objeto	❌ No, el hilo sigue manteniendo el bloqueo
Pausa el hilo actual?	✅ Sí	✅ Sí
Se puede despertar con notify() o notifyAll()?	✅ Sí, se usa para coordinar hilos	❌ No, solo se despierta cuando el tiempo expira
Uso típico	Comunicación entre hilos (Productor-Consumidor)	Pausar un hilo sin afectar otros procesos

notify() y notifyAll()

Junto con **wait**(); **notify**() y **notifyAll**() se utiliza para la comunicación entre hilos. Cuando un hilo entra en un estado de espera a través de una llamada a **wait**() estará esperando a que otro subproceso llame a un método **notify**() o **notifyAll**() en el mismo objeto.

Al igual que wait, **estos métodos deben ser invocados dentro de un método o bloque synchronized** que controle el monitor del objeto con el que se está trabajando.

notify() y notifyAll()

- **notify**(): una llamada a este método activa un único subproceso que está esperando en el monitor de este objeto. Si hay subprocesos esperando a este objeto, se elige uno de ellos para despertarlo. La elección es arbitraria.
- **notifyAll**(): una llamada a este método activa todos los subprocesos que están esperando en el monitor de este objeto. Un hilo espera en el monitor de un objeto llamando a uno de los métodos de espera. Los subprocesos despertados no podrán continuar hasta que el subproceso actual renuncie (libere voluntariamente) el bloqueo de este objeto.

Implementación

```
public synchronized void produce(int value) {
        while (availabe) {
                try {
                        wait();
                } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                }
        }
        data = value;
        availabe = true;
        System.out.println("Productor produjo: " + valor);
        notify();
}
```

```
public synchronized int consume() {
        while (!availabe) {
                try {
                        wait();
                } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                }
        }
        available = false;
        System.out.println("Consumidor consumió: " + data);
        notify();
        return data;
}
```

¿Por qué un while antes del wait()?

```
while (availabe) {
        try {
                wait();
        } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
        }
}
notify();
```

```
if (availabe) {
        try {
                wait();
        } catch (InterruptedException e) {
                Thread.currentThread().interrupt()
        }
}
notify();
```

Despertar espurio

El **despertar espurio** ("**spurious wakeup**") ocurre cuando un hilo que estaba en wait() es despertado sin haber recibido una señal de **notify**() o **notifyAll**().

Esto puede suceder debido a:

- Implementaciones internas del sistema operativo y la JVM.
- Optimizaciones en la planificación de hilos.
- Interrupciones inesperadas.

Cómo evitar el despertar espurio

La solución recomendada es usar un ciclo while en lugar de un if al verificar la condición antes de llamar a wait().

```
while (availabe) {
        try {
                wait();
        } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
        }
}
notify();
```

Si el hilo se despierta sin que la condición se haya cumplido, el while vuelve a ejecutar wait(), evitando errores. En cambio, con if, el hilo continuaría sin volver a verificar la condición.

Grado en Ingeniería Informática - 3er Curso

Concurrencia y distribución

Martín Pérez Pérez (martiperez@uvigo.es)
Noelia García Hervella (nghervella@esei.uvigo.es)
Guillermo Blanco González (guillermo.blanco.gonzalez@uvigo.gal)