Programación de Servicios y Procesos

PRÁCTICA 2

Iago Otero
ESICA | 2019/2020

1. Explica el concepto de región crítica.

La región crítica es aquella sección de código en la que, si no se controla y entran distintos hilos al mismo tiempo, el valor que intentan leer o modificar puede no ser el real ya que si por ejemplo en una clase tenemos un atributo contador que lo que hace es aumentar cada vez que se accede a él, este aumentaría con un orden desordenado, por lo tanto esta sección debe estar controlada para evitar que varios hilos entren en ese fragmento de código.
Para controlar ese “problema” se necesita un mecanismo de sincronización en la entrada y salida de la sección crítica para asegurarnos de que nadie más está accediendo a esa porción de código, se puede utilizar por ejemplo un semáforo, monitores, el algoritmo de Dekker y Peterson, etc.
Se debe tomar como instrucción atómica, es decir, debe cumplir dos condiciones, la primera es que no se debe notificar el cambio del código en cuestión hasta que este haya terminado,
y la segunda es que si por algún motivo ese código no se puede modificar al menos a un paso intermedio este tendrá que volver a su estado original, es decir, que o se realiza o no se realiza, no puede quedar mal hecho o a medias.

Esta región crítica puede implementarse de tres formas.
Exclusión mutua: es equivalente a la atomicidad de operaciones. Cuando un hilo accede a la región crítica el resto debe esperar a que este lo notifique para así poder acceder de nuevo a esta sección y así sucesivamente.
Comportamiento idéntico: este algoritmo obliga a que, aunque los distintos hilos accedan a una misma región crítica a que el resto de hilos actúen igual.
Comportamiento transaccional: este algoritmo verifica que los pasos que se van a realizar van a llegar a un resultado válido, por lo que se ejecutará, sino, en caso contrario se notificará de igual modo y no se realizará el proceso.

2. ¿Qué es y cómo funciona una instrucción TAS (Test-And-Set)?

Una instrucción TAS es una instrucción que se usa para escribir un valor en una variable y además devuelve el valor anterior de forma atómica, una operación que se tiene que realizar al completo o no se realiza.
Si varios hilos intentan acceder a la misma variable y en ese momento se encuentra realizando esa operación no se realizará, hasta que esta termine.
El hilo que intente acceder sabrá que si el valor es X (el valor que se escogerá para que se reemplace (set)) a la hora de estar utilizándolo, este esperará a que sea distinto para poder entrar.

3. Explica el Principio de la Bandera para comprobar la exclusión mutua. ¿Qué es? ¿Cómo funciona?

Es un teorema que nos permite comprobar la propiedad de exclusión mutua en un fragmento de código. Este teorema se basa en que antes de que cada hilo intente entrar en la sección crítica levanten una bandera (por ejemplo, poner una variable a true), y a continuación fijarse en la variable de los distintos hilos que están intentando acceder a ese fragmento de código, al verse unos a los otros con la bandera levantada (la cual significa que va a entrar en dicho código) estos deberán esperar a que los otros la bajen para poder acceder al fragmento de código.

Este teorema se utiliza para comprobar que solo un hilo o proceso puede obtener acceso a la sección crítica.

4. ¿Qué es la Espera Activa en programación concurrente?

La espera activa es una técnica donde un hilo comprueba repetidamente una condición, y esto ocurre se realiza por ejemplo en el teorema de las banderas, ya que los distintos hilos que tiene la bandera levantada están “esperando” (en realidad están comprobando continuamente a un cambio) para poder entrar en esa sección.

5. ¿Qué es la Condición de Carrera en programación concurrente?

La condición de Carrera es un “problema” a la hora de ejecutar un procedimiento en programación concurrente ya que si distintos hilos ejecutan este procedimiento podría haber una variación en el resultado de los cálculos.
Esto es debido a que cuando un proceso está esperando a una secuencia de eventos arbitrarios que van a trabajar sobre un mismo recurso compartido estos se ven comprometidos compitiendo en carrera por llegar antes que otro, de manera que según llegue uno u otro primero, el resultado será distinto. Estas condiciones no solo se dan a nivel de proceso si no a nivel de sistema.
Por ello se utilizan monitores en Java los cuales nos ayudan a resolver este tipo de incidencias.

6. ¿Para qué sirve el modificador "volatile" en Java?

Volatile es un modificador que se aplica en algunos atributos para avisarle al compilador de que varios hilos accederán a ese atributo cada vez que intenten acceder a ese atributo tendrán que renovar el valor en el momento de usarlo ya que cada hilo puede contener esa variable en caché, por lo tanto, si otro hilo ya accedió a ese valor y lo modificó este puede que aún contenga un valor erróneo por lo que deberá volver a recoger ese valor para tratarlo.

7. ¿Qué es un Monitor en programación concurrente? ¿Para qué sirve?

Un monitor es una clase destinada a ser usada por distintos hilos con la certeza de que no habrá errores, es decir, un monitor controlará el proceso de sincronización entre los distintos hilos de ejecución. Disponemos de distintos métodos para la implementación de tareas concurrentes, por ejemplo, el uso de la palabra reservada synchronized la cual indicará que los distintos hilos deberán acceder a dicho método de forma síncrona.

8. El paquete "java.util.concurrent" proporciona clases con herramientas útiles para la programación concurrente. Define para qué sirven las siguientes:

    a. **Clase Future.**
    Es una clase abstracta que presenta métodos para verificar si el cálculo se ha completado, esperar su finalización y recuperar el resultado del cálculo.

    b. **Clase CountDownLatch.**
    Es una clase que permite que uno o más subprocesos esperen a que se realice el conjunto de operaciones en otros subprocesos.

    c. **Clase ExecutorService.**
    Esta clase proporciona métodos para gestionar la terminación de métodos que pueden producir un Future para seguir el progreso de una o más tareas asíncronas.

    d. **Clase Semaphore.**
    Esta clase se usa para restringir el número de hilos que pueden acceder a algún recurso.

    e. **Clase BlockingQueue.**
    Implementa un método Queue que admite operaciones de espera en cola para su ejecución.

    f. **Clase AtomicInteger**.
    Esta clase permite declarar int los cuales funcional a nivel atómico, de forma que sólo se puede usar por un único acceso para ser leído o modificado.