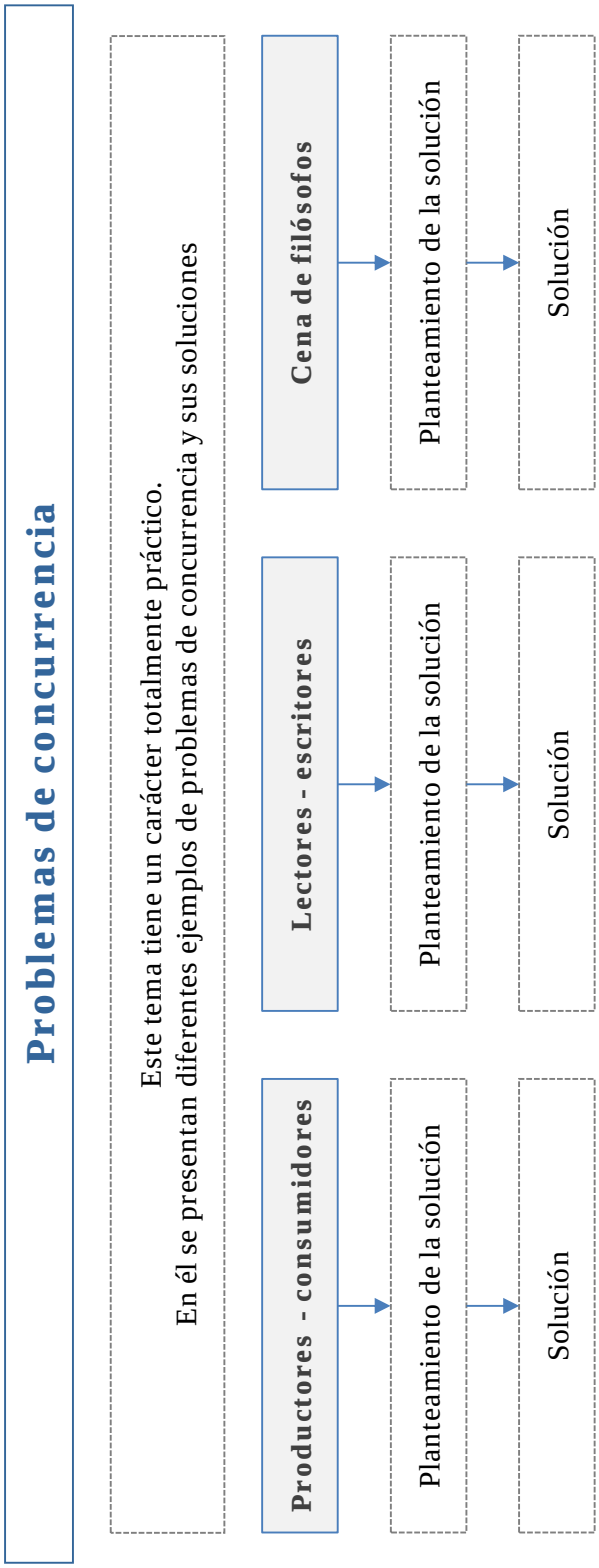
Problemas comunes dentro de la programación concurrente

TEMA 10

Esquema



Ideas clave

10.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las **Ideas clave** que encontrarás a continuación.

En este tema vamos a ver la implementación en Java de algunos de los problemas de concurrencia más representativos utilizando monitores.

10.2. El problema de los productores consumidores

Problema

Uno o más productores generan cierto tipo de datos y los sitúan en un *buffer*. Un único consumidor saca elementos del *buffer* de uno en uno. El sistema debe impedir la superposición de operaciones sobre el *buffer*. Es decir, solo un proceso (productor o consumidor) puede acceder al *buffer* en un instante dado.

Planteamiento de la solución

- Para que un consumidor pueda acceder al *buffer* necesita que exista algún dato en él. Para tener en cuenta está restricción utilizaremos una variable booleana `valueSet`, que indicará si el buffer contiene o no algún dato.

- Para que un productor pueda acceder al *buffer* necesita que no exista ningún dato en él. Para tener en cuenta está restricción utilizaremos una variable booleana `valueSet`.

- Para que consumidor y productor no accedan a la vez al buffer utilizaremos un monitor, es decir, declararemos los métodos de la clase que implementa el buffer (en nuestro caso la clase Q), como `synchronized`.

Solución

Consulta la solución en el aula virtual.

10.3. El problema de los lectores y escritores

Problema

Este problema modeliza el **acceso a grandes bases de datos**. Un objeto de datos (como un fichero o registro) tiene que compartirse entre varios procesos concurrentes. Alguno de estos procesos puede que únicamente desee leer el contenido del objeto compartido, mientras que otros puede que deseen actualizarlo (leer y escribir).

Distinguimos dos tipos de procesos:

Lectores	Escritores
Solo están interesados en la lectura	Leen y escriben

Si dos lectores acceden al objeto de datos compartido simultáneamente, no se producen efectos adversos. Sin embargo, si un escritor y otro proceso (lector o escritor) acceden pueden surgir problemas

Planteamiento de la solución

- El objeto compartido se implementa en la clase `Documento`.
- Debido a que un escritor no puede acceder al recurso al mismo tiempo que cualquier otro proceso (lector o escritor), debemos garantizar que los escritores tengan acceso exclusivo al objeto compartido.
- En la clase `Documento` utilizaremos los atributos lectores y escritores, para mantener el número de lectores y escritores que están accediendo al documento.
- En el método `entrar_lector` se comprobará que el número de escritores (atributo escritores) que está accediendo al recurso compartido es 0, para garantizar el acceso exclusivo de los escritores.
- En el método `entrar_escritor` se comprobará que el número de lectores (atributo lectores) y el número de escritores (atributo escritores) es cero, para garantizar el acceso exclusivo de los escritores.

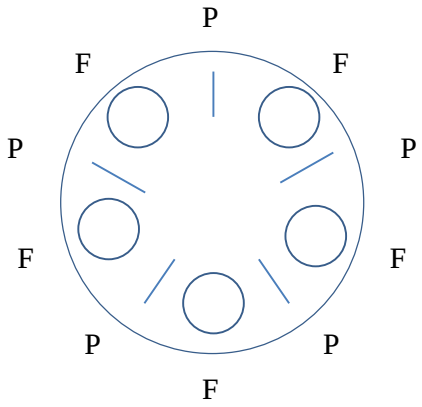
Solución

Consulta la solución en el aula virtual.

10.4. La comida de los filósofos

Problema

Cinco filósofos dedican sus vidas a pensar y comer. Los filósofos comparten una mesa circular rodeada por cinco sillas, cada una de las cuáles pertenece a un filósofo. En la mesa hay cinco platos de arroz y cinco palillos.



Cuando un filósofo piensa, no se relaciona con sus colegas. De vez en cuando, un filósofo siente hambre y trata de coger los dos palillos que están más cerca de él. Un filósofo puede tomar solamente un palillo cada vez; obviamente no puede coger un palillo que ya se encuentra en la mano de un vecino. Cuando un filósofo tiene sus dos palillos simultáneamente, come sin dejar los palillos. Cuando ha acabado de comer, vuelve a dejar los dos palillos y empieza a pensar de nuevo.

Se trata de establecer un sistema por el cual todos los filósofos pueden comer y que lo hagan cuando tengan hambre, es decir, con el mayor grado de concurrencia posible.

Planteamiento de la solución

Un filósofo podrá comer cuando el filósofo de su izquierda y el filósofo de su derecha no lo estén haciendo. Si uno de ellos está comiendo, deberá esperar. Para ello, en el método `permisoComer`, de la clase `Mesa`, se comprobará el estado de los filósofos que están al lado del que desea comer.

Solución

Consulta la solución en el aula virtual.

Lo + recomendado

No dejes de leer...

Problema de agentes y fumadores

Este artículo plantea dos soluciones al problema de los agentes y fumadores, mediante monitores y `locks`.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: http://resolviendoejerciciosdeprogramacion.blogspot.com.es/2014/05/ejercicio-de-concurrencia-en-java.html

+ Información

A fondo

Filósofos comensales

Solución al problema de los filósofos comensales utilizando la clase `Semaphore` de Java.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
https://netsis.es/cena-filosofos-java/

Barbero durmiente

Solución al problema del barbero durmiente utilizando la clase `Semaphore` de Java.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://sleepingbarber.googlecode.com/svn-history/r8/trunk/sleepingbarber/java/BarberoDurmiente.java

Test

1. En el problema de los filósofos comensales, ¿qué problema se daría si todos los filósofos cogiesen al mismo tiempo el palillo que se encuentra a su derecha?

A. Inanición.

B. Interbloqueo.

C. No se garantiza la exclusión mutua.

D. Apropiación de recursos compartidos.

2. En la solución propuesta al problema de los filósofos comensales, ¿para qué se realiza una llamada a `notifyAll` en el método `cedePermiso`?

A. Para bloquear el acceso al objeto compartido.

B. La llamada al método `notifyAll` no es necesaria.

C. Para avisar al resto de filósofos, que están esperando, de la liberación del objeto compartido.

D. Ninguna de las anteriores es correcta.

3. En el problema de los lectores/escritores, ¿qué situación podríamos evitar si el número de escritores fuese 1?

A. Lecturas y escrituras simultáneas.

B. Escrituras simultáneas.

C. Lecturas simultáneas.

D. Ninguna de las anteriores.

4. En la solución propuesta al problema de los productores/consumidores, ¿qué sucedería si no utilizásemos la variable `valueSet`?

A. No habría exclusión mutua.

B. No podríamos garantizar que cuando un consumidor accede al *buffer* hay datos en él.

C. Los productores sufrirían inanición.

D. Ninguna de las anteriores.

5. En el problema de los productores/consumidores, ¿cuántos consumidores pueden acceder simultáneamente al objeto compartido?:

A. Solo uno.

B. Todos los que haya.

C. Solo uno en el caso de que un productor también esté accediendo al recurso compartido.

D. Ninguna de las anteriores.

6. En la solución propuesta al problema de los lectores/escritores, ¿cuándo se despertará un hilo lector que ha realizado una llamada a `wait`?

A. Cuando un lector haga la llamada a `notify`.

B. Cuando un escritor haga la llamada `sleep`.

C. Cuando un escritor haga la llamada a `notify`.

D. Ninguna de las anteriores.

7. En la solución propuesta al problema de los filósofos comensales, ¿para qué se utiliza la variable `private boolean [] comiendo`?

A. Para indicar si los filósofos de la mesa están comiendo o no.

B. Para indicar si los filósofos quieren abandonar la mesa.

C. Para indicar si los filósofos han terminado de comer.

D. Ninguna de las anteriores.

8. En el problema de los filósofos comensales, ¿cuáles es el objeto compartido por el que compiten los filósofos?

A. El arroz.

B. Las sillas.

C. La mesa.

D. Los palillos.

9. En el problema de los filósofos comensales, si en la mesa hubiese cinco filósofos, ¿cuál sería el mayor grado de concurrencia posible?

A. Tres filósofos comiendo.

B. Un filósofo comiendo.

C. Dos filósofos comiendo.

D. Ninguna de las anteriores.

10. En el problema de los lectores/escritores, si hubiese cinco lectores y cinco escritores, ¿cuál sería el mayor grado de concurrencia posible?

A. Cinco lectores y cinco escritores.

B. Cinco lectores.

C. Cinco lectores y un escritor.

D. Ninguna de las anteriores.