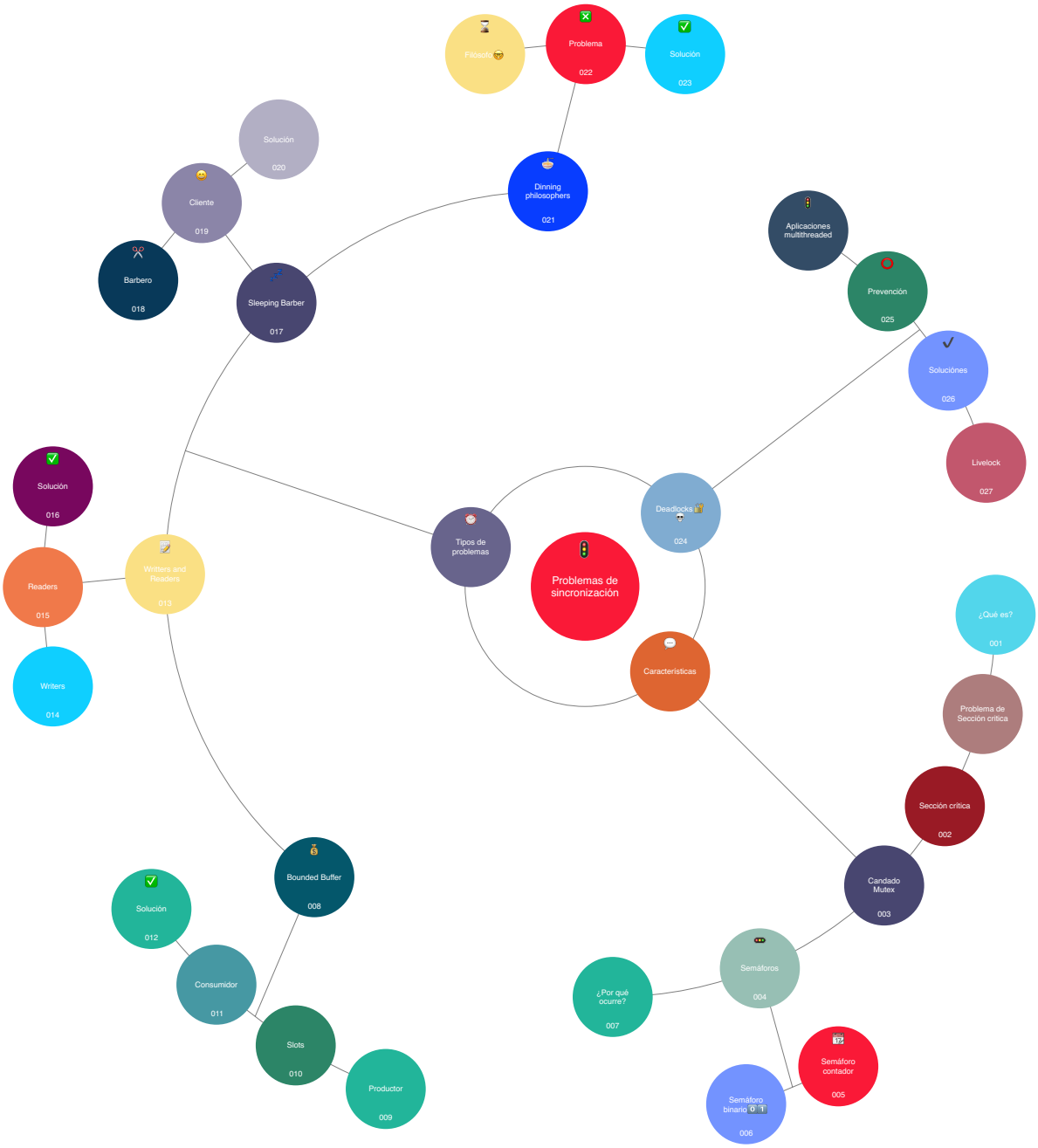
08/05/2020 17:04

Problemas de sincronización



001 ¿Qué es?

Procesos que comparten datos o código, que pueden ser interrumpidos en cualquier momento.

002 Sección crítica

Parte del programa donde se accede a un recurso compartido que solo puede ser modificado o leído por un proceso o thread a la vez.
El problema ocurre cuando se comparten variables, archivos, tablas o otros recursos que pueden quedar en un estado inconsistente.

003 Candado Mutex

Variable bolea a que indica si un bloqueo está disponible o no.

004 Semáforos

Provee un método más sofisticado que el candado mutex para sincronización.
Consiste en dos llamadas, p(wait), y V(signal).
El semáforo es un valor entero que es modificado por wait y signal.

005 Semáforo contador

Valor entero puede recorre un dominio sin restricciones.

006 Semáforo binario 0 1

El valor es entre 0 y 1 como el candado mutex.

007 ¿Por qué ocurre?

El acceso concurrente de datos compartidos puede generar inconsistencias.

008 Bounded Buffer

El problema del productor y consumidor está relacionado con el acceso concurrente a un recurso.

009 Productor

El productor es aquel que crea objetos o datos y debe bloquearse cuando el buffer está lleno.

010 Slots

Lugar de memoria donde se guarda información en un bounded buffer.

011 Consumidor

Es aquel que elimina los objetos o datos del buffer y debe bloquearse cuando el buffer está vacío.

012 Solución

Se pueden utilizar semáforos binarios para poder sincronizar a los consumidores y productores.
Una posible solución es utilizar 2 semáforos.
1er semáforo(empty):
-Se inicializa en N (max capacidad del arreglo)
-Productor espera a que el semáforo le dé permiso de escribir en el buffer.
-El consumidor libera el semáforo después de leer información en el buffer.
2do semáforo(data):
-Inicializa en 0
-Consumidor espera a que el semáforo le de permiso de leer info en el buffer. -Productor libera el semáforo una vez haya terminado de escribir

013 Writters and Readers

Problema de escritores y lectores.

014 Writers

Son procesos que acceden en modo escritura al mismo recurso.

015 Readers

Son procesos que acceden a un recurso en modo lectura.

016 Solución

• Utilizando un semáforo binario inicializado a 1 podemos conseguir que solo un proceso del tipo escritor pueda en cada momento escribir en el recurso como protocolo de acceso en modo escritura tendremos que usar wait() en el semáforo y como protocolo de salida haremos signal() sobre el mismo semáforo.
• Así nos evitamos que los lectores puedan acceder al recurso cuando un escritor lo esté utilizando.

• Utilizando un semáforo binario inicializado a 1 podemos conseguir que solo un proceso del tipo escritor pueda en cada momento escribir en el recurso como protocolo de acceso en modo escritura tendremos que usar wait() en el semáforo y como protocolo de salida haremos signal() sobre el mismo semáforo.
• Así nos evitamos que los lectores puedan acceder al recurso cuando un escritor lo esté utilizando.

017 Sleeping Barber

Problema del barbero durmiente.

018 Barbero

La analogía es de un barbero que tiene su tienda, en su tienda hay una silla para cortar el cabello y tiene n sillas para la sala de espera.
Si no hay clientes, el barbero duerme en su propia silla donde corta el cabello.

019 Cliente

Cuando un cliente llega, el tiene que despertar al barbero para que lo atienda.

Si hay muchos clientes y el barbero está cortando el cabello a algún cliente, los clientes restantes tienen que pasar a la sala de espera si es que hay sillas disponibles, si no los cliente se van.

020 Solución

En este caso se utilizan tres semáforos: clientes, que cuenta los clientes
en espera, barberos, si el barbero está ocupado (0 o 1), y mutex, que se utiliza para la exclusión mutua. También se necesita una variable, la espera, que también cuenta los clientes en espera. La razón de tener espera es que no hay forma de leer el valor actual de un semáforo.

•En esta solución, un cliente que entra en la tienda tiene que contar el número de clientes en espera. Si es menor que el número de sillas, se queda; si no, se va.
• Cuando el barbero se presenta a trabajar por la mañana, ejecuta el procedimiento de barbero, causando que bloquee al cliente del semáforo porque inicialmente es 0. Permanece dormido hasta que aparece el primer cliente.
• Cuando llega un cliente, lo ejecuta, comenzando por adquirir el mutex para entrar en una región crítica. Si otro cliente entra poco después, el segundo no podrá hacer nada hasta que el primero haya liberado el mutex. El cliente entonces comprueba si el número de clientes que esperan es menor que el número de sillas. Si no, libera el mutex y se va sin corte de pelo. Si hay una silla disponible, el cliente incrementa la variable entera, esperando. Luego hace un Up en los clientes del semáforo, despertando así al barbero.
• En este punto, el cliente y el barbero están ambos despiertos. Cuando el cliente suelta el mutex, el barbero lo agarra, hace algunas tareas domésticas, y comienza el corte de pelo.
• Cuando el corte de pelo termina, el cliente sale del procedimiento y deja la tienda. A diferencia de nuestros ejemplos anteriores, no hay un bucle para el cliente porque cada uno tiene sólo un corte de pelo. El peluquero hace un bucle, sin embargo, para tratar de conseguir el siguiente cliente. Si uno está presente, se le da un corte de pelo. Si no, el barbero se duerme.

021 Dinning philosophers

Problema de la cena de los filósofos.

022 Problema

Existen cinco filósofos cuya misión es pensar y comer, pero no al mismo tiempo. El problema radica en que se dispone de una mesa con cinco platos y cinco palillos situado entre cada plato. Para comer, un filósofo necesita dos palillos: uno situado a la derecha del plato y el otro a la izquierda, con el condicionante de que un filósofo no puede tomar un palillo si este ha sido tomado por otro hasta que este último lo deje libre.

023 Solución

Poner 3 status: HAMBRIENTO, COMIENDO, PENSANDO. Dar prioridad para comer al HAMBRIENTO.
• Dar turnos para que cada filosofo cheque los palillos a ambos lados. Checar si la persona de al lado esta comiendo.

024 Deadlocks 🔐 💀

Un thread en espera puede nunca cambiar su estado, porque los recursos que ha solicitado están siendo ocupados por otros threads. A esto se le llamada Deadlock.

025 Prevención

Exclusión mutua.
Sostener y esperar.
Que no haya derecho preferente.
Espera circular.

026 Soluciónes

Ignorar el problema y pretender que los deadlocks nunca ocurrieron.
Usar un protocolo que prevenga o evite deadlocks, asegurando que el sistema nunca ingrese a un estado deadlock.
Permitir que el sistema entre a un estado deadlock, detectarlo, y recuperarse.

027 Livelock

Otra forma de falla, similar al deadlock; ambos prevén dos o más threads a proceder.
Este en especial ocurre cuando un thread continuamente intenta una acción que falla.