Clase 21/10/2020

El tiempo es una referencia que se utiliza para establecer un orden en una secuencia de eventos.

Como vimos anteriormente, si dos computadoras no interactúan entonces no es necesario que sus relojes estén sincronizados.

Por otra parte, si dos computadoras interactúan, en general no es importante que coincidan en el tiempo real sino en el orden en que ocurren los eventos.

Happens-before

En el artículo Time, Clocks, and the Ordering of Events in Distributed Systems (1978) Leslie Lamport define la relación A→B (se lee, A happens-before B) de la siguiente manera:

1. Si A y B son eventos del mismo proceso y A ocurre antes que B, entonces A → B
2. Si A es el envío de un mensaje y B la recepción del mensaje, entonces A → B

La relación happens-before tiene las siguientes propiedades:

Transitiva: Si A → B y B → C entonces A → C
Anti-simétrica: Si A → B entonces no(B → A)
Irreflexiva: no(A → A) para cada evento A

Relojes lógicos

Se define un **reloj lógico Ci** para un procesador Pi como una función Ci(A) la cual asigna un número al evento A.

Un reloj lógico se implementa como un contador sin una relación directa con un reloj físico, como es el caso de los contadores de "ticks" de las computadoras digitales.

Dados los eventos A y B, si el evento A ocurre antes que el evento B, entonces Ci(A)<Ci(B), por tanto:

Si A → B entonces Ci(A)<Ci(B)

Esto significa que si A happens-before B, entonces el evento A ocurre en un

tiempo lógico menor al tiempo lógico en que ocurre el evento B.

Algoritmo de sincronización de relojes lógicos de Lamport

Ahora utilizaremos la relación happends-before definida por Lamport para sincronizar relojes lógicos en diferentes procesadores (computadoras).

Supongamos que tenemos los procesadores P1, P2 y P3. Cada procesador tiene un reloj lógico (contador) que se incrementa periódicamente mediante un thread.

El reloj lógico del procesador P1 se incrementa en 6, el reloj lógico del procesador P2 se incrementa en 8 y el reloj lógico del procesador P3 se incrementa en 10.

¿Cómo sincronizar los relojes lógicos de los tres procesadores de manera que los eventos que ocurren en los procesadores puedan ordenarse?

Lamport resuelve este problema utilizando la relación happends-before para sincronizar los relojes lógicos de diferentes procesadores. Explicaremos el algoritmo de sincronización de relojes lógicos de Lamport con un ejemplo.

Supongamos que al tiempo **6** el procesador P1 envía el mensaje m1 al procesador P2, este procesador recibe el mensaje al tiempo **16**. Al tiempo **24** el procesador P2 envía el mensaje m2 al procesador P3, este procesador recibe el mensaje al tiempo **40**.

Hasta ahora todo es correcto, debido a que el mensaje m1 es enviado al tiempo 6 y recibido al tiempo 16, y el mensaje m2 es enviado al tiempo 24 y recibido al tiempo 40, es decir, el tiempo de envío es menor al tiempo de recepción.

Al tiempo **60** el procesador P3 envía el mensaje m3 al procesador P2, este procesador recibe el mensaje al tiempo **56**. Lo anterior contradice la definición de la relación happends-before, ya que la recepción de un mensaje debe ocurrir después del envío del mismo mensaje.

Entonces lo que se hace es ajustar el reloj lógico del procesador P2, asignando el tiempo lógico del procesador P3 cuando envía el mensaje m3 más uno, es decir, se modifica el reloj lógico del procesador P2 a 61 cuando recibe el mensaje m3.

Al tiempo 69, el procesador P2 envía el mensaje m4 al procesador P1, este procesador recibe el mensaje al tiempo 54, lo cual contradice la relación happends-before.

Entonces se aplica el mismo procedimiento para el ajuste del reloj lógico del

procesador P1, por tanto el reloj lógico de este procesador se modifica 70 cuando recibe el mensaje m4.

Exclusión mutua

Ahora veremos algunos algoritmos que resuelven el problema de exclusión mutua cuando dos o más procesadores requieren acceder simultáneamente un recurso compartido (impresora, memoria, CPU, disco, etc.).

Existen dos tipos de bloqueos, los **bloqueos exclusivos** y **bloqueos compartidos**. Dependiendo del recurso en particular, se puede utilizar solo bloqueos exclusivos o bien, bloqueos exclusivos y compartidos.

Si un procesador bloquea un recurso de manera exclusiva, ningún procesador puede bloquear el recurso de manera exclusiva o compartida.

Si un procesador bloquea un recurso de manera compartida, otros procesadores pueden bloquear el mismo recurso de manera compartida, es decir, múltiples bloqueos compartidos sobre el mismo recurso pueden co-existir.

Si un recurso tiene uno o más bloqueos compartidos, ningún procesador puede obtener un bloqueo exclusivo sobre el mismo recurso.

Los bloqueos exclusivos pueden utilizarse para controlar, por ejemplo, el uso de impresoras. Para el acceso a dispositivos de almacenamiento (memorias, discos, etc.), los bloqueos exclusivos se utilizan para proteger operaciones de escritura, mientras que los bloqueos compartidos se utilizan para proteger lecturas.

Por ejemplo, un bloqueo compartido sobre un archivo en el disco permite que varios procesadores puedan leer el archivo, pero no permite que ningún procesador escriba el archivo. Por otra parte, un bloqueo exclusivo sobre el archivo garantiza que solo un procesador pueda escribir y ningún otro procesador pueda leer o escribir el archivo.

En un sistema distribuido las computadoras compiten por adquirir el bloqueo sobre un recurso. En una situación de competencia existe la posibilidad de que una o varias computadoras nunca adquieran el recurso debido a deficiencias en el algoritmo de exclusión. Cuando una computadora no puede adquirir un bloqueo se dice que se presenta **inanición**.

Otro problema que se puede presentar en un algoritmo de exclusión es el **inter-bloqueo** (*dead-lock*). El inter-bloqueo es una situación en la que dos o más

procesadores esperan la liberación de un bloqueo, sin que está liberación se pueda realizar.

Para ilustrar una situación de inter-bloqueo, supongamos dos procesadores P1 y P2 que acceden a dos recursos R1 y R2. Por simplicidad asumimos que los bloqueos sobre los recursos son exclusivos.

Podemos ver que el procesador P1 requiere bloquear el recurso R1, debido a que R1 está desbloqueado el procesador P1 adquiere el recurso R1. De la misma manera el procesador P2 requiere bloquear el recurso R2, debido a que R2 está desbloqueado el procesador P2 adquiere el recurso R2.

Cuando el procesador P1 requiere bloquear el recurso R2, no puede hacerlo ya que el procesador P2 lo tiene bloqueado, por tanto queda esperando a que R2 se libere. Así mismo, cuando el procesador P2 requiere bloquear el recurso R1, no puede hacerlo ya que el procesador P1 lo tiene bloqueado. Por lo tanto, ambos procesadores quedan bloqueados permanentemente.

Para evitar que los procesos se bloqueen, los manejadores de bases de datos (p.e. MySQL) detectan el inter-bloqueo como un error, de manera que los programadores puedan controlar la situación.

Algoritmo centralizado para exclusión mutua

Veremos un algoritmo centralizado para implementar la exclusión mutua en un sistema distribuido.

Primeramente necesitamos un nodo que haga las funciones de coordinador, este nodo deberá tener una cola dónde se formaran los nodos que esperan por el recurso.

Explicaremos el algoritmo con un ejemplo. Supongamos que tenemos cuatro nodos. El nodo 3 hace las funciones de coordinador. En un momento dado, el nodo 1 envía una petición al coordinador, debido a que el recurso esta desbloqueado, el coordinador regresa al nodo 1 el mensaje "OK", lo que significa que el nodo 1 ha adquirido el recurso.
Después el nodo 2 envía una petición al coordinador, como el recurso está bloqueado por el nodo 1 el coordinador agrega el nodo 2 a la cola de espera; mientras tanto el nodo 2 queda esperando la respuesta del coordinador.

Cuando el nodo 1 desbloquea el recurso, envía un mensaje de liberación al coordinador, el coordinador extrae el primer nodo de la cola de espera, en este caso el nodo 2, y envía el mensaje "OK" al nodo 2. Entonces el nodo 2 continua

con la ejecución de su proceso.

Algoritmo distribuido para exclusión mutua

La desventaja del algoritmo centralizado es que el coordinador puede saturarse si recibe muchas peticiones, por esta razón es mejor implementar un algoritmo descentralizado.

Ricart y Agrawala (1981) desarrollaron el siguiente algoritmo distribuido para exclusión mutua, el cual permite controlar el acceso a múltiples recursos:

- Cuando un nodo requiere acceso al recurso, envía un mensaje de petición a todos los nodos (incluso a sí mismo) y espera hasta recibir el mensaje "OK" de cada nodo. El mensaje incluye el ID del recurso, el número de nodo y el tiempo lógico.
- Cuando un nodo recibe el mensaje de petición:
  - Si el nodo no esta esperando por el recurso envía un mensaje "OK" al emisor del mensaje de petición.
  - Si el nodo posee el recurso, coloca en la cola de espera el número de nodo que viene en el mensaje de petición.
  - Si el nodo está esperando por el recurso, compara el tiempo lógico T1 del mensaje de petición que recibió con el tiempo lógico T2 del mensaje de petición que previamente envió. Si T1<T2 entonces envía el mensaje "OK" al nodo que envío el mensaje de petición. Si T2<T1, entonces coloca en la cola de espera el número de nodo del mensaje de petición recibido.
- Cuando el nodo libera el recurso:
  - Extrae todos los nodos de la cola de espera y envía el mensaje "OK" a los nodos extraídos de la cola.

El algoritmo anterior requiere que los nodos cuenten con relojes lógicos sincronizados (algoritmo de Lamport) y que cada nodo cuente con **una cola de espera para cada recurso** (recordar que el mensaje de petición incluye el ID del recurso).

Ilustraremos el algoritmo con el siguiente ejemplo. Supongamos que tenemos tres nodos, cada nodo tiene una cola de espera.

El nodo 0 requiere acceso a un recurso, por tanto envía un mensaje de petición a todos los nodos; el mensaje incluye el tiempo lógico 8. Al mismo tiempo el nodo 2 requiere acceso al mismo recurso, entonces el nodo 2 envía un mensaje de petición a todos los nodos, incluyendo el tiempo lógico 12.
El nodo 1 recibe los mensajes de petición de los nodos 0 y 2, debido a que el nodo

1 no está esperando por el recurso, envía sendos mensajes "OK" a los nodos 0 y 2.

El nodo 0 recibe el mensaje de petición que envió el nodo 2. Compara el tiempo lógico T1 del mensaje de petición que recibió con el tiempo lógico T2 del mensaje de petición que previamente envió, en este caso T1=12 y T2=8. Como T2<T1 coloca el nodo 2 en la cola de espera.

El nodo 2 recibe el mensaje de petición que envió el nodo 0. Compara el tiempo lógico T1 del mensaje de petición que recibió con el tiempo lógico T2 del mensaje de petición que previamente envió, en este caso T1=8 y T2=12. Como T1<T2 entonces envía el mensaje "OK" al nodo 0.

Debido a que el nodo 0 recibió "OK" de todos los nodos, adquiere el recurso. Cuando el nodo 0 desbloquea el recurso extrae el primer nodo de la cola de espera, en este caso el nodo 2, y envía el mensaje "OK" al nodo 2, entonces el nodo adquiere el recurso.

Cuando el nodo 2 desbloquea el recurso revisa si tiene algún nodo en la cola de espera, si es así extrae el nodo de la cola y le envía el mensaje "OK". En este caso no hay nodos esperando en la cola, por tanto el nodo 2 continua su proceso sin más.

Algoritmo de token en anillo (token ring)

El algoritmo de token en anillo permite implementar la exclusión mutua en un sistema distribuido de manera muy simple, sin embargo tiene la desventaja de que requiere tener una conexión estable entre los nodos, por lo tanto, este algoritmo generalmente se implementa utilizando conexiones físicas.

Supongamos que tenemos ocho nodos conectados en una topología de anillo.

El nodo 0 envía un token (un dato) al nodo 1, el nodo 1 envía el token al nodo 2, y así sucesivamente.

El algoritmo de exclusión mutua utilizando un token en anillo es el siguiente:

- Cuándo un nodo requiere acceso al recurso compartido:
  - Espera recibir el token.
  - El nodo adquiere el bloqueo cuando recibe el token.
  - Cuando el nodo desbloquea el recurso, envía el token al siguiente nodo.
- Si un nodo no requiere acceso al recurso, simplemente pasa el token al siguiente nodo en el anillo.