SO - 1P - Ejercicios de Sync (4, 5)

4

Recuerdo las deficiones de las propiedades listadas

Supongo que CRIT es `recorrerCamino()`

- *LOCK-FREEDOM* $\equiv \Box(\#TRY \geq 1 \land \#CRIT = 0 \Rightarrow \Diamond \#CRIT > 0)$

  Si hay al menos un proceso intentando de entrar a la seccion critica y no hay ninguno en ella, entonces en algun momento del futuro habra algun proceso dentro de la seccion critica.

  Hago el caso de `VehiculoSubiendo` y el otro es análogo.

  Si hay algun `VehiculoSubiendo` en TRY necesariamente debe ser en la primera linea, `nadieBajando.wait()`. Como no hay ninguno en la seccion critica, quiere decir que o bien el resto de los `VehiculoSubiendo` todavia no entro a CRIT, por lo que le deberian haber hecho `signal` o justo el ultimo `VehiculoBajando` esta a punto de hacer el `nadieSubiendo.signal()`, por lo que tambien deberia pasar.

  En ambos casos, el proceso llega a CRIT.

- Y usando predicados auxiliares,

  - *Lograr entrar* $IN(i) \equiv i \in TRY \Rightarrow \Diamond i \in CRIT$
  - *Salir* $OUT(i) \equiv i \in CRIT \Rightarrow \Diamond i \in REM$
  - *STARVATION-FREEDOM* $\equiv \forall i \Box OUT(i) \Rightarrow \forall i \Box IN(i)$

  Intuitivamente, si todos salen, todos entran.

  Tomando como CRIT a `recorrerCamino`,

  Esto se cumple, ya que si todos salen de la seccion critica, en particular tambien sale el que hace que `cantSubiendo.getAndDec()` sea 0, con lo que se liberaria el turnstile de los que van en la direccion opuesta, por el cual pasarian todos.

- CRASH-FREEDOM **no** se cumple

  `nadieBajando` y `nadieSubiendo` comienzan en 1, entonces cualquier proceso que le haga `wait` pasa de largo a la primera, a menos que alguien lo blockee con un `wait`.

  Podria suceder entonces que un `VehiculoSubiendo` entre, haga `wait` en `nadieBajando`, pase de largo porque arrancaba en 1, y justo antes del if lo desaloje el scheduler.

  En este punto, entra un `VehiculoBajando`, analogamente pasa de largo por `nadieSubiendo` y a partir de aqui ambos procesos continuan en cualquier orden, entrando a la vez a CRIT y rompiendo CRASH-FREEDOM.

  Cuando le hagan wait a `nadieSubiendo` y `nadieBajando`, el semaforo va a quedar en 0, pero alguien ya paso por ahi, entonces ambos haran `recorrerCamino` a la vez.

5

Es necesario garantizar la propiedad que no se cumple, CRASH-FREEDOM.

El problema surge de que puede ser que se bloquee el turnstile despues de que haya pasado alguien.

En principio la idea seria mover el bloqueo arriba del wait. Pero si se hace sin mas, podria entrar uno de cada direccion a la vez, hacer los dos wait en `nadieSubiendo` y `nadieBajando`, asi bloqueandose mutuamente y causando un deadlock.

Para evitar esto, agrego un mutex.

Es importante notar que cuando el primer `VehiculoSubiendo` haga `nadieSubiendo.wait()` con el lock tomado, nunca pueden bloquearse. Esto es porque el valor de ese semaforo es siempre 1, excepto cuando este proceso le hace `wait`. Analogamente, sucede lo mismo con `VehiculoBajando`

Ademas, cuando hagan `nadieBajando.wait()`, si se quedan bloqueados, es porque ya hay vehiculos bajando, y eventualmente se liberara, no se dara un deadlock porque al bajar no es necesario tomar el lock.

```
// Variables compartidas
atomic<int> cantSubiendo = 0;
atomic<int> cantBajando = 0;
Mutex mutex;    // podria ser Semaforo(1)
Semaforo nadieBajando(1);
Semaforo nadieSubiendo(1);

VehículoSubiendo {
    mutex.lock()
        if (cantSubiendo.addAndGet(1) == 1) {
            // No había nadie subiendo por el camino
            // Bloqueo el turnstile para que nadie pueda bajar
            nadieSubiendo.wait();
        }

        // Si habia alguien bajando, no estaran con este mutex tomado.
        nadieBajando.wait();
    mutex.unlock()

    // Hago pasar al resto
    nadieBajando.signal();

    recorrerCamino();

    if (cantSubiendo.decAndGet(1) == 0) {
        // Ya no queda nadie subiendo
        // Libero el turnstile para que otros puedan bajar
        nadieSubiendo.signal();
    }
}

// Es simetrica pero la agrego por completitud
VehículoBajando {
    mutex.lock()
        if (cantBajando.addAndGet(1) == 1) {
            // No había nadie bajando por el camino
            // Bloqueo el turnstile para que nadie pueda subir
            nadieBajando.wait();
        }

        // Si habia alguien subiendo, no estaran con este mutex tomado.
        nadieSubiendo.wait();
    mutex.unlock()

    // Hago pasar al resto
    nadieSubiendo.signal();

    recorrerCamino();

    if (cantBajando.decAndGet(1) == 0) {
        // Ya no queda nadie bajando
        // Libero el turnstile para que otros puedan subir
        nadieBajando.signal();
    }
}
```

Se debe cumplir CRASH-FREEDOM, pues no puede suceder que un proceso pase cuando el otro le quiso bloquear el turnstile, ya que el acto de pasar por el turnstile y el bloqueo del mismo se hacen "dentro" del mutex.