Práctica 2. Programación Paralela

Sergio Rodrigo Angulo

Como primera idea podemos desarrollar un monitor y un invariante que se cumpla a lo largo del programa y que esté diseñado de manera que dos coches de distintas direcciones o un coche y un peatón no estén al mismo tiempo en el puente y así, poder evitar accidentes.

class Monitor
{I}
coches_norte: int = 0
coches_sur: int = 0
peatones: int = 0

coches_norte_esperando: int = 0
coches_sur_esperando: int = 0
peatones_esperando: int = 0

cocheN: Variable Condición
cocheS: Variable Condición
ped: Variable Condición.
{I}

I = Invariante
coches_norte ≥ 0, coches_sur ≥ 0, peatones ≥ 0
coches_norteesperando ≥ 0, coches_sur ≥ 0, peatones_esperando ≥ 0
· coches_norte > 0 ⇒ coches_sur = 0 ∧ peatones = 0
· coches_sur > 0 ⇒ coches_norte = 0 ∧ peatones = 0
· peatones > 0 ⇒ coches_norte = 0 ∧ coches_sur = 0.

Observamos que según hemos diseñado el invariante del monitor es imposible que haya coche de sentido opuesto o coches y peatones al mismo tiempo, por tanto vemos que el puente es seguro.
Definimos ahora las funciones para que los coches y peatones entren y salgan del puente.

- wants_enter_car(direction)
  if direction == NORTH:
    coches_norte_esperando += 1.
    cocheN.wait(coches_sur == 0 ∧ peatones == 0)
    coches_norte_esperando -= 1
    coches_norte += 1

  if direction == SOUTH
    coches_sur_esperando += 1.
    cocheS.wait(coches_norte == 0 ∧ peatones == 0)
    coches_sur_esperando -= 1
    coches_norte += 1.

- leaves_car (direction)

```
if direction == NORTH
    coches_norte -= 1
    if coches_norte == 0
        cocheS.notify_all()
        ped.notify_all()
if direction == SOUTH
    coches_sur -= 1
    if coches_sur == 0
        cocheN.notify_all()
        ped.notify_all()
```

- Wats_enter_pedestrian

```
peatones_esperando += 1
ped.wait(coches_norte == 0 ∧ coches_sur == 0)
peatones_esperando -= 1
peatones += 1.
```

- leaves_pedestrian

```
peatones -= 1
if peatones == 0
    cocheS.notify_all()
    cocheN.notify_all().
```

- car (direction)

```
loop
    monitor.Wants_enter_car(direction)
    monitor.leaves_car.(direction)
```

- pedestrian

```
loop
    monitor.Wants_enter_pedestrian()
    monitor.leaves_pedestrian().
```

Esta solución es coherente en cuanto a circulación y gracias a los wait en las variables condición no crea deadlocks. El único problema que hay es que si, por ejemplo, tenemos un coche en el sur y un gran número de coches en el norte, suponiendo que ya ha entrado un coche del norte en el puente el coche del sur tendrá que esperar a que pasen todos los del norte.

Para resolver este problema de inanición crearemos una variable turnos en el monitor t.q.

$$\begin{cases} turno == -1 & \text{no hay nadie esperando} \\ turno == 0 & \text{coches del norte pueden pasar} \\ turno == 1 & \text{coches del sur pueden pasar} \\ turno == 2 & \text{peatones pueden pasar.} \end{cases}$$

Además habrá que añadir al invariante $turno \in \{-1, 0, 1, 2\}$.

Este sistema de turnos funcionará de manera que cuando un coche de una dirección (o peatón) entre en el puente podrán seguir entrando elementos del mismo tipo hasta que ese primer coche salga. En ese momento el turno cambiará votando de norte a peatones de peatones a sur y de sur a norte.

Observamos, además, que aunque cambie el turno los vehículos esperarán a que se vacíe el puente para iniciar su turno así aunque haya un número de ~~coches~~ elementos muy grandes en alguna de las direcciones, se irán alternando de manera que todos puedan pasar.

Estos cambios están realizados directamente en el programa.