Germán Sánchez Costa

DESARROLLO PRÁCTICA 2 PRPA

Comenzamos dando una solución sencilla que sepamos que asegura la seguridad. Para esto definiremos tres values en el init y tres condiciones, una por cada objeto que tiene que cruzar. Los values servirán para contar la cantidad de objetos de un mismo tipo que están cruzando en ese momento. Usaremos un wait_for para dar paso con las condiciones, llamando a funciones que verifiquen si hay objetos "distintos" cruzando. Quedaría entonces:

INV
- ncar1crossing.value ≥ 0
- ncar2crossing.value ≥ 0
- npedcrossing.value ≥ 0
- npedcrossing.value ≥ 1 → ncar2crossing.value = 0
- ncar1crossing.value ≥ 1 → npedcrossing = 0 ∧ ncar2crossing = 0
- ncar2crossing.value ≥ 1 → ncar1crossing = 0 ∧ npedcrossing = 0

```
wants_enter_ped()
pedcross.wait_for(cross_ped)
npedcrossing += 1
```

```
leaves_ped()
npedcrossing -= 1
car1cross.notify_all()
car2cross.notify_all()
```

```
wants_enter_car(dir)
if dir == 0:
    car2cross.
    car1cross.wait_for(cross_1)
    ncar1crossing += 1
else:
    car2cross.wait_for(cross_2)
    ncar2crossing += 1
```

```
leaves_car(dir)
  if dir == 0:
    ncar1crossing -= 1
    car2cross.notify_all()
    pedcross.notify_all()
  else:
    ncar2crossing -= 1
    car1cross.notify_all()
    pedcross.notify_all()
```

```
cross_1()
  return ncar2crossing == 0 ∧ npedcrossing == 0
cross_ped()
  return ncar2crossing == 0 ∧ ncar1crossing == 0
```

Con esto nos aseguramos que nunca entre un objeto distinto siempre y cuando haya otro cruzando. Sin embargo, esta implementación tiene un problema. En caso de que se encadenen entradas de un mismo objeto, digamos por ejemplo, coches dirección sur, en ningún momento podrán pasar coches dirección norte o peatones, quedándose en espera hasta el final y generado inanición. Necesitamos una forma de dar paso a otros objetos si hay muchos esperando. Para esto, vamos a añadir al init 3 values para contar la cantidad de coches o peatones esperando y un cuarto para que haga de sistema de "turnos". De esta forma, en caso de que la cantidad de, por ejemplo, peatones esperando sea mayor que un número (5, mismamente), cambiamos a su turno y no dejamos que dejen pasar más coches. El invariante será el invariante anterior más:

$0 \leq$ nped waiting ≤ 5
$0 \leq$ ncar1 waiting ≤ 5
$0 \leq$ ncar2 waiting ≤ 5

nped waiting $\geq 5 \rightarrow$ turn = 0
ncar1 waiting $\geq 5 \rightarrow$ turn = 1
ncar2 waiting $\geq 5 \rightarrow$ turn = 2

] INV 2

Y los cambios serían añadir los contes de nped waiting - ncar1 waiting y ncar2 waiting y cambiar las returns de las funciones que recibe el wait_for para que verifiquen que el turno es el correcto.

Con esto se soluciona el problema de la inanición, pero aparece otro. Como el cambio de turno se realiza sólo cuando el número de objetos esperando es 5, tenemos que si hemos acabado de generar todos y hay, por ejemplo, 3 peatones aún esperando sin ser su turno, nunca pasarán, ya que no se cambiará nunca a su turno. Este es un problema de deadlock y se tiene que solucionar, ya que si no los procesos no acabarán nunca. Para ello, simplemente introduciremos un cuarto valor para los turnos: -1. El value turno tendrá este valor siempre que no haya más de 5 objetos esperando

de ningún tipo. Si tenemos el turno neutro, puede pasar cualquier objeto - verificando siempre la seguridad. Así nunca se nos quedarán procesos colgados esperando a su propio turno. Con esto tendríamos ya nuestra solución final, segura y que carece de inanición y deadlocks