

PRACTICA 2 : PUENTE DE ANBITE

Vamos a desarrollar este problema poco a poco.

A) caso base: que cumpla la condición de seguridad.

caso base: que cumplamos
En este caso nos centraremos únicamente en que no haya ninguna situación
peligrosa dentro del puente: " "

Monitor - Puente Ambite

```

Monitor = Puentes/Carros =
{
    ncar0 : int = 0 # número de coches en el puente en sentido 0 (NORTH)
    ncar1 : int = 0 # " " " " " " " " " " 1 (SOUTH).
    nped : int = 0 # " " peatones " " "
    mutex : lock() # semáforo binario
    pasa - car0 : VC = TRUE
    pasa - car1 : VC = TRUE
    pasa - ped : VC = TRUE
}

```

$$\left[\begin{array}{l} n_{car0} \geq 0 \wedge n_{car1} \geq 0 \wedge n_{ped} \geq 0 \\ n_{ped} > 0 \Rightarrow n_{car0} = n_{car1} = 0 \\ n_{car0} > 0 \Rightarrow n_{car1} = n_{ped} = 0 \\ n_{car1} > 0 \Rightarrow n_{car0} = n_{ped} = 0 \end{array} \right] \text{ INV.}$$

```
def prede-passar-car0 (self) → bool:
    return nped == 0 and ncar1 == 0
```

```
def puelle - pasar - car1 (self) → bool:
    return nped == 0 and ncar0 == 0
```

```
def prede-passar-ped (self) → bool:
    return n card == 0 and n card1 == 0
```

```
def coche_entra():
    mutex.wait()
    if direccion_coche == 0:
        pasa_car0.wait_for(puede_pasar_car0)
        ncar0 += 1
    elif direccion_coche == 1:
        pasa_car1.wait_for(puede_pasar_car1)
        ncar1 += 1
    mutex.signal()
```

```
def coche-sale():
    mutex.wait()
    if direccion-coche == 0:
        ncar0 -= 1
        if ncar0 == 0:
            puede-pasar-car1.notify-all()
            puede-pasar-ped.notify-all()
    else:
        ncar1 -= 1
        if ncar1 == 0:
            puede-pasar-ped.notify-all()
            puede-pasar-car0.notify-all()
    mutex.signal()
```

```

def ped-entra()
    mutex.wait()
    pasar-ped.wait-for(puede-pasar-ped)
    nped += 1
    mutex.signal()

```

```

def ped-sale()
    mutex.wait
    nped -= 1
    if nped == 0:
        puede-pasar-car0.notify-all()
        puede-pasar-car1.notify-all()

```

③ En el código anterior verifica gracias a las variables condición la seguridad del puente, puesto que siempre se espera a que no haya ninguno de los otros en sus casos; es decir, nunca va a ocurrir que coincidan dentro del puente los car0 con un car1 ni peatones.

④ También se asegura la exclusión mutua gracias al semáforo binario "mutex" colocados al inicio y al final de cada acción.

B) caso mejorado: vamos a mejorar el código anterior para asegurar la ausencia de deadlocks y la ausencia de inanición.
 Para ello, definiremos, además de las que teníamos antes, las siguientes variables y modificaremos las funciones:

Monitor - Puente Ambiente:

@# las variables de antes

ncar0-waiting : int = 0

ncar1-waiting : int = 0

nped-waiting : int = 0

turno : int = 0

$\left\{ \begin{array}{l} \text{turno} == 0 \rightarrow \text{turno de car0} \\ \text{turno} == 1 \rightarrow \text{" " car1} \\ \text{turno} == 2 \rightarrow \text{" " ped} \end{array} \right.$

def puede-pasar-car0():

return ncar1 == 0 and nped == 0 and (turn == 0 or (nped-waiting == 0 and ncar1-waiting == 0))

def puede-pasar-car1():

return ncar0 == 0 and nped == 0 and (turn == 1 or (nped-waiting == 0 and ncar0-waiting == 0))

def puede-pasar-ped():

return ncar0 == 0 and ncar1 == 0 and (turn == 2 and (ncar0-waiting == 0 and ncar1-waiting == 0))

~~44~~ # el invariante anterior

$$\left[ncar_waiting \geq 0 \wedge ncar1_waiting \geq 0 \wedge nped_waiting \geq 0 \right] INV$$

```
def entra-coche ()
    mutex.wait ()
    if direccion-coche == 0:
        ncar0-waiting += 1
        pasa-car0.wait-for (puede-pasar-car0)
        ncar0-waiting -= 1
        ncar0 += 1
    else:
        ncar1-waiting += 1
        pasa-car1.wait-for (puede-pasar-car1)
        ncar1-waiting -= 1
        ncar1 += 1
    mutex.signal
```

```
def sale-coche ()
    mutex.wait ()
    if direccion-coche == 0:
        ncar0 -= 1
        turno = 1
        if ncar == 0:
            puede-pasar-car0.notify-all ()
            puede-pasar-car1.notify-all ()
            puede-pasar-ped.notify-all ()
        else:
            ncar1 -= 1
            turno = 2
            if ncar1 == 0:
                puede-pasar-car0.notify-all ()
                puede-pasar-car1.notify-all ()
                puede-pasar-ped.notify-all ()
    mutex.signal ()
```

```
def entra-ped ():
    mutex.wait ()
    nped-waiting += 1
    pasa-ped.wait-for (puede-pasar-ped)
    nped-waiting -= 1
    nped += 1
    mutex.signal ()
```

```

def sale_ped():
    mutex.wait()
    nped -= 1
    turno = 0
    if nped == 0:
        puede_pasar_car0.notify_all()
        puede_pasar_car1.notify_all()
        puede_pasar_ped.notify_all()
    mutex.signal()

```

⊙ Como en este código no hemos cambiado las condiciones que aseguraban la seguridad del puente, la seguimos teniendo.

En las variables condición que se evalúan para entrar al puente, se asegura que el número de coches en sentido contrario y peatones sea cero, (en cada caso según corresponde). Es por eso que nunca podrán producirse "choques entre coches" ni "atropellos a peatones".

NOTA: Por como se ha planteado el problema, el turno se pasa cuando el primer coche o peatón que entró en cierta dirección sale del puente. De modo que, si mientras está en el puente otro igual que él quiere entrar podría hacerlo. Y, cuando el primero pasa su turno, (si hay otro esperando) se bloquea para que sigan entrando los iguales a él. Así, en algún momento saldrá el último en entrar, se notificará a los VC y podrán entrar los demás.

⊙ Aseguremos que no hay deadlocks:

En este problema, un deadlock podría suceder si hay dos (o más) tipos de pasajeros diferentes (coches 0, coches 1 o peatones) que quieran entrar al puente y ninguno de ellos entrase.


Esta situación no puede ocurrir, primero porque existe una jerarquía establecida por los turnos, si todos estuviesen esperando a entrar, uno de ellos tendría el turno y entraría. Si ocurriese que dos quieren entrar y ninguno de ellos tiene el turno, habría entrado el primero que hubiese querido entrar, ya que en la evaluación de la VC sería ninguno en el puente and (su turno es ninguno de los otros espera), como hemos dicho, no sería su turno, pero por ser el primero en esperar, se cumpliría la otra condición y podría entrar.

Así, confirmamos la ausencia de deadlocks.

⊙ Ausencia de inanición:

Esto está completamente asegurado gracias a los turnos.

Se podría dar una situación de inanición cuando hubiese un proceso que nunca entrase al puente. Supongamos que este fuese coches0.

Por como está diseñado el código, los turnos se pasan de nos a otros circularmente, es decir, . Supongamos que entran coches1 al puente, como en algún momento el primer coche que entró saldrá del puente, se pasará el turno a los peatones y, cuando el primer peatón entre, también habrá un momento en el que tenga que salir. De este modo, pasarán el turno a coches0, en caso de que en ese momento no hubiese ningún coche esperando, se pasará al final a coches0, coches1 o peatones, según quién quisiese entrar primero, y así es claro que no hay inanición.