Práctica 2

- En primer lugar propondremos una solución básica en la que únicamente se garantiza la seguridad del puente, es decir, dos coches en direcciones distintas no pueden pasar a la vez por el puente, y si un peatón está pasando por el puente, entonces ningún coche puede pasar hasta que no salga el peatón.

  - Monitor

    coches_norte : int = 0
    coches_sur : int = 0
    peatones : int = 0

    N_puede_pasar : VC
    S_puede_pasar : VC
    P_puede_pasar : VC

- El invariante viene dado por:
  - coches_norte ≥ 0, coches_sur ≥ 0, peatones ≥ 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

- wants_enter_car(direction):

```
mutex.wait()
if direction == SOUTH:
    S_puede_pasar.wait_for(coches_norte==0 ^ peatones==0)
    coches_sur += 1
else:
    N_puede_pasar.wait_for(coches_sur==0 ^ peatones==0)
    coches_norte += 1
mutex.signal()
```

- leaves_car(direction):

```
mutex.wait()
if direction == SOUTH:
    coches_sur -= 1
    if coches_sur == 0:
        N_pede_pasar.notify_all()
        P_pede_pasar.notify_all()
else:
    coches_norte -= 1
    if coches_norte == 0:
        S_pede_pasar.notify_all()
        P_pede_pasar.notify_all()
mutex.signal()
```

- wants_enter_pedestrian():

```
mutex.wait()
P_pede_pasar.wait_for(coches_norte == 0 ^ coches_sur == 0)
peatones += 1
mutex.signal()
```

- leaves_pedestrian():

```
mutex.wait()
peatones -= 1
if peatones == 0:
    N_pede_pasar.notify_all()
    S_pede_pasar.notify_all()
mutex.signal()
```

- En esta versión se generan los problemas de inanición y deadlocks
- El problema de inanición se genera porque si consideramos el caso en el que un coche del norte cruza el puente, y llega otro, y así sucesivamente, aunque vengan coches de la otra dirección queriendo pasar, hasta que no acaben de pasar todos los coches del norte no podrán pasar. Si hay una cantidad finita de coches y peatones entonces al final todos acabarán pasando, pero si hay por ejemplo una cantidad arbitrariamente grande de coches del norte, entonces sí podría generarse el problema de inanición.

- Una posible modificación de la versión básica sería introducir las variables esperando_N: int = 0, esperando_S: int = 0 y esperando_P: int = 0 que llevan la cuenta del número de coches del norte, sur y peatones que están esperando para cruzar el puente respectivamente. Además introducimos una nueva condición en los wait_for de wants_enter_car y wants_enter_pedestrian:

  - wants_enter_car(direction):

    if direction == SOUTH:
        S_puede_pasar.wait_for(coches_norte == 0 ^ peatones == 0 ^ (esperando_N ≤ 10) ^ (esperando_P ≤ 4))
    else:
        N_puede_pasar.wait_for(coches_sur == 0 ^ peatones == 0 ^ ^(esperando_S ≤ 10) ^ (esperando_P ≤ 4))

  - wants_enter_pedestrian():

    P_puede_pasar.wait_for(coches_norte == 0 ^ coches_sur == 0 ^ (esperando_N ≤ ^ (esperando_S ≤ 10))

- Con la introducción de los wait conseguimos evitar que si por ejemplo se generan infinitos coches del norte y estos son los primeros en pasar, si los coches del sur superan el límite de 10 esperando (condición añadida en el wait_for) entonces no permitirá que sigan circulando los del norte.

- El problema de lo anterior es que se genera deadlock, pues se podría dar el caso de que estén pasando coches del norte y tengan 15 coches del sur y 15 peatones. Por las condiciones impuestas en los wait_for, los coches del norte no podrían pasar, pero tampoco los del sur pues esperando_N = 15 ≯ 4 y tampoco peatones pues esperando_S = 15 ≯ 10. Por tanto se tendría en el caso en el que todos están esperando y ninguno puede pasar.

- Para poder solucionar esto se añade la variable turno: int = -1, de manera que asignemos un valor distinto para los coches del norte, sur y peatones y así se puedan ir turnando entre todos para ir pasando por el puente. Sólo falta modificar leaves_car(direction) y leaves_pedestrian():

- leaves_car(direction):

```
:
if direction == SOUTH:
    :
    if turno == (turno de los coches del sur)
        if esperando_N != 0:
            turno = (turno de los coches del norte)
        elif esperando_P != 0
            turno = (turno de los peatones)
        else:
            turno = -1
    ...
else:
    if turno == (turno de los coches del norte):
        if esperando_S != 0:
            turno = (turno coches del sur)
        elif esperando_P != 0:
            turno = (turno peatones)
        else: turno = -1
```

- llegan_peatones():

```
...
if turno == (turno peatones):
    if esperando_N != 0:
        turno = (turno coches norte)
    elif esperando_S != 0:
        turno = (turno coches sur)
    else:
        turno = -1
...
```

- Si el turno = -1, implica que el puente está vacío.
- En lo anterior lo que se ha hecho es elegir una preferencia a la hora de ceder los turnos. La idea es que si estan pasando los coches del norte por ejemplo, en cuanto sale el primer coche del puente, si hay coches del sur esperando se les cede el turno a ellos, y sino, a los peatones
- Finalmente, para poder garantizar la ausencia de deadlocks, basta con añadir en las condiciones de las variables condición para que puedan pasar los coches o peatones, si turno correspondiente o que el puente esté vacío, es decir, en los wait for se hacen las siguientes modificaciones:
  - S_puede_pasar: wait_for(coches_norte == 0 ^ peatones == 0 ^ [(esperando_N ≤ 10) ^ (esperando_P ≤ 4)) V turno = (turno coches sur) V turno = -1])
  - N_puede_pasar: wait_for(coches_sur == 0 ^ peatones == 0 ^ [(esperando_S ≤ 10) ^ (esperando_P ≤ 4)) V turno = (turno coches norte) V turno = -1])
  - P_puede_pasar: wait_for((coches_norte == 0 ^ coches_sur == 0 ^ [(esperando_N ≤ 10) ^ (esperando_S ≤ 10)) V turno = (turno peatones) V turno = -1])
- La versión resultante de añadir todas modificaciones anteriores no contiene inanición ni deadlocks pues siempre se garantiza que todos puedan pasar (gracias a los turnos) y que se sigan manteniendo las seguridad del puente.
