Práctica 2. Puente de Ambite

Para cumplir la seguridad se tiene que cumplir siempre que: no haya coches y peatones a la vez en el puente y que no haya coches en sentidos opuestos.

Luego el invariante, (escrito de manera informal) es:

- nº coches norte $\geq 0 \wedge$ nº coches sur $\geq 0 \wedge$ nº de peatones ≥ 0
- nº coches norte esperando $\geq 0 \wedge$ nº coches sur esperando $\geq 0 \wedge$ nº peatones esperando ≥ 0
- nº coches norte $> 0 \Rightarrow$ nº coches sur $= 0 \wedge$ peatones $= 0$
- nº coches sur $> 0 \Rightarrow$ nº coches norte $= 0 \wedge$ peatones $= 0$
- nº peatones $> 0 \Rightarrow$ nº coches norte $= 0 \wedge$ nº coches sur $= 0$

En la siguiente hoja he escrito el código sencillo y básico que cumple la seguridad, y le he añadido en rosa el código necesario para evitar la inanición y en azul el código necesario para evitar deadlocks.
La explicación es la siguiente:

En la versión inicial se garantiza la seguridad, ya que hemos creado las variables y funciones necesarias para asegurar lo que hemos escrito en el invariante (utilizamos una condición para que los coches del sur puedan pasar, otra para los del norte puedan pasar y otra para que los peatones puedan pasar).
Sin embargo, con ese código puede dase el caso de que solo entre un tipo (cochesN, cochesS o peatón) y que los otros dos tengan que esperar a que crucen todos. Es decir que vayan entrando por bloques y no unos pocos de cada cosa intercalándose. De hecho aquí porque hemos decidido que el nº de vehículos y peatones es finito, pero si fuera infinito podría ocurrir que el puente estuviera siempre ocupado por un tipo y nunca pudieran entrar los otros vehículos.
Para evitar estos problemas de inanición, añadimos unas variables que nos indican cuántos coches y peatones están esperando.

Aún así, pueden haber problemas de deadlocks si por ejemplo hay coches del norte y del sur esperando a la vez, y como no queda indicado cuál entra en el puente, se quedarán todos esperando indefinidamente. Para solucionar esto introducimos turnos, es decir, establecemos un orden de preferencias y añadiremos una variable que se va modificando según eso.

Leyenda:
Si nos quedamos solo con el código que no está subrayado tenemos el código básico inicial que cumple la seguridad.

En rosa para evitar inanición

En azul para evitar deadlocks

```
"""
Solution to the one-way tunnel
"""
import time
import random
from multiprocessing import Lock, Condition, Process
from multiprocessing import Value

SOUTH = 1
NORTH = 0

NCARS = 15
NPED = 5
TIME_CARS_NORTH = 0.5  # a new car enters each 0.5s
TIME_CARS_SOUTH = 0.5  # a new car enters each 0.5s
TIME_PED = 5 # a new pedestrian enters each 5s
TIME_IN_BRIDGE_CARS = (1, 0.5) # normal 1s, 0.5s
TIME_IN_BRIDGE_PEDESTRIAN = (3, 0.5) # normal 3s, 0.5s

class Monitor():
    def __init__(self):
        self.mutex = Lock()
        self.patata = Value('i', 0)


        self.ncoches_N = Value('i', 0)
        self.ncoches_S = Value('i', 0) #
        self.npeatones = Value('i', 0) #


        self.esperando_coches_N = Value('i', 0)
        self.esperando_coches_S = Value('i', 0)
        self.esperando_peatones = Value('i', 0)


        self.pasan_coches_N = Condition(self.mutex)
        self.pasan_coches_S = Condition(self.mutex)
        self.pasan_peatones = Condition(self.mutex)



        self.turn = Value('i', -1)


    def car_N_can_pass(self):
        return (self.ncoches_S.value==0 and
self.npeatones.value==0) and (self.turn==2  or
(self.esperando_coches_S.value <= 5 and
self.esperando_peatones.value<=3) or self.turn==-1)

    def car_S_can_pass(self):
        return (self.ncoches_N.value==0 and
self.npeatones.value==0) and (self.turn==1 or
(self.esperando_coches_N.value <= 5  and
self.esperando_peatones.value<=3) or self.turn==-1)
```

```
    def pedestrian_can_pass(self):
        return (self.ncoches_N.value==0 and
self.ncoches_S.value==0) and (self.turn==0 or
(self.esperando_coches_N.value <= 5 or
self.esperando_coches_S.value <= 5) or self.turn==-1)


    def wants_enter_car(self, direction: int) -> None:
        self.mutex.acquire()
        self.patata.value += 1

        if direction == NORTH:
            self.esperando_coches_N.value +=1
            self.pasan_coches_N.wait_for(self.car_N_can_pass)
            self.esperando_coches_N.value -=1

            if self.turn.value == -1:
                self.turn.value = 2
            self.ncoches_N.value += 1
        else:
            self.esperando_coches_S.value +=1
            self.pasan_coches_S.wait_for(self.car_S_can_pass)
            self.esperando_coches_S.value -=1

            if self.turn.value == -1:
                self.turn.value = 1
            self.ncoches_S.value += 1


        self.mutex.release()

    def leaves_car(self, direction: int) -> None:
        self.mutex.acquire()
        self.patata.value += 1


        if direction == NORTH:
            self.ncoches_N.value -= 1

            if self.turn.value == 2:


                if self.esperando_peatones.value != 0:
                    self.turn.value = 0
                elif self.esperando_coches_S.value != 0:
                    self.turn.value = 1
                else:
                    self.turn.value = -1



            if self.ncoches_N.value == 0:
                self.pasan_coches_S.notify_all()
                self.pasan_peatones.notify_all()

        else:
```

```
        self.ncoches_S.value -= 1

        if self.turn.value == 1:

            if self.esperando_coches_N.value != 0:
                self.turn.value = 2
            elif self.esperando_peatones.value != 0:
                self.turn.value = 0
            else:
                self.turn.value = -1

        if self.ncoches_S.value == 0:
            self.pasan_coches_N.notify_all()
            self.pasan_peatones.notify_all()


    self.mutex.release()


def wants_enter_pedestrian(self) -> None:
    self.mutex.acquire()
    self.patata.value += 1



    self.esperando_peatones.value += 1
    self.pasan_peatones.wait_for(self.pedestrian_can_pass)
    self.esperando_peatones.value -= 1

    if self.turn.value == -1:
        self.turn.value = 0
    self.npeatones.value += 1


    self.mutex.release()

def leaves_pedestrian(self) -> None:
    self.mutex.acquire()
    self.patata.value += 1


    self.npeatones.value -= 1

    if self.turn.value == 0:

        if self.esperando_coches_S.value != 0:
            self.turn.value = 1
        elif self.esperando_coches_N.value != 0:
            self.turn.value = 2
        else:
            self.turn.value = -1

    if self.npeatones.value == 0:
        self.pasan_coches_S.notify_all()
        self.pasan_coches_N.notify_all()
```

```
        self.mutex.release()

    def __repr__(self) -> str:
        return f'Monitor: {self.patata.value}'



def delay_car_north() -> None:
    valor=random.normalvariate(TIME_IN_BRIDGE_CARS[0],
TIME_IN_BRIDGE_CARS[1])
    if valor<0:
        valor=0
    time.sleep(valor)

def delay_car_south() -> None:
    valor=random.normalvariate(TIME_IN_BRIDGE_CARS[0],
TIME_IN_BRIDGE_CARS[1])
    if valor<0:
        valor=0
    time.sleep(valor)

def delay_pedestrian() -> None:
    valor=random.normalvariate(TIME_IN_BRIDGE_PEDESTRIAN[0],
TIME_IN_BRIDGE_PEDESTRIAN[1])
    if valor<0:
        valor=0
    time.sleep(valor)


def car(cid: int, direction: int, monitor: Monitor)  -> None:
    print(f"car {cid} heading {direction} wants to enter.
{monitor}")
    monitor.wants_enter_car(direction)
    print(f"car {cid} heading {direction} enters the bridge.
{monitor}")
    if direction==NORTH :
        delay_car_north()
    else:
        delay_car_south()
    print(f"car {cid} heading {direction} leaving the bridge.
{monitor}")
    monitor.leaves_car(direction)
    print(f"car {cid} heading {direction} out of the bridge.
{monitor}")

def pedestrian(pid: int, monitor: Monitor) -> None:
    print(f"pedestrian {pid} wants to enter. {monitor}")
    monitor.wants_enter_pedestrian()
    print(f"pedestrian {pid} enters the bridge. {monitor}")
    delay_pedestrian()
    print(f"pedestrian {pid} leaving the bridge. {monitor}")
    monitor.leaves_pedestrian()
    print(f"pedestrian {pid} out of the bridge. {monitor}")
```

```
def gen_pedestrian(monitor: Monitor) -> None:
    pid = 0
    plst = []
    for _ in range(NPED):
        pid += 1
        p = Process(target=pedestrian, args=(pid, monitor))
        p.start()
        plst.append(p)
        time.sleep(random.expovariate(1/TIME_PED))

    for p in plst:
        p.join()

def gen_cars(direction: int, time_cars, monitor: Monitor) ->
None:
    cid = 0
    plst = []
    for _ in range(NCARS):
        cid += 1
        p = Process(target=car, args=(cid, direction,
monitor))
        p.start()
        plst.append(p)
        time.sleep(random.expovariate(1/time_cars))

    for p in plst:
        p.join()

def main():
    monitor = Monitor()
    gcars_north = Process(target=gen_cars, args=(NORTH,
TIME_CARS_NORTH, monitor))
    gcars_south = Process(target=gen_cars, args=(SOUTH,
TIME_CARS_SOUTH, monitor))
    gped = Process(target=gen_pedestrian, args=(monitor,))
    gcars_north.start()
    gcars_south.start()
    gped.start()
    gcars_north.join()
    gcars_south.join()
    gped.join()


if __name__ == '__main__':
    main()
```