PRÁCTICA 2

Mª del Mar Lafuente

Partiremos de una solución sencilla que garantice la seguridad en el puente, (esto es, no hay coches y peatones a la vez en el puente y tampoco habrá simultáneamente coches en sentido contrario):

```
monitor Puente                    (Sabiendo que Norte = 0 y Sur = 1)
    integer ncars_0 = 0  # coches que van hacia el norte
    integer ncars_1 = 0  # coches que van hacia el sur
    integer npedestrians = 0   # peatones
    condition can_cars_0, can_cars_1, can_pedestrians


    operation wants_enter_car(direction)   # direction==0 (Norte), direction==1 (sur)
            if direction == 0 :
                can_cars_0.waitC(npedestrians == 0 ∧ ncars_1 == 0)
                ncars_0 += 1
```

```
    else :   # direction == 1
        can-cars-1 . waitC ( npedestrians == 0  ∧  ncars_0 == 0)
        ncars-1  += 1


operation  Leaves_car (direction) :
    if  direction == 0 :
        ncars-0  -=  1
        if  ncars-0 == 0 :
            signalC ( can-cars-1)
            signalC ( can-pedestrians)
    else:
        n-cars-1  -= 1
        if  n_cars-1  == 0 :
            signalC ( can-pedestrians )
            signalC ( can-cars-0 )


operation  wants-enter-pedestrian( ) :
    can-pedestrians . waitC ( ncars-0 == 0  ∧  ncars-1 == 0)
```

```
            npedestrians += 1


    operation leaves_pedestrian():
            npedestrians -= 1
            if npedestrians == 0:
                signalC(ncars_0)
                signalC(ncars_1)
```

car (direction)	pedestrian ()
loop	loop
Puente.wants_enter_car(direction)	Puente.wants_enter_pedestrian()
crosses	crosses
Puente.leaves_car(direction)	Puente.leaves_pedestrian()

- Con la anterior solución garantizamos la seguridad en el puente, pues la variables condición que deben cumplirse antes de entrar al puentes nos dicen que para poder pasar, por ejemplo, un coche dirección norte, NO debe haber en el puente ni peatones ni coches dirección sur. Análogamente se da el resto de casos. □

$$\overset{npedestrians}{\overset{\parallel}{}}\qquad \overset{ncars\_0}{\overset{\parallel}{}}\qquad \overset{ncars\_1}{\overset{\parallel}{}}$$

- Sea P = peatones, N = coches dirección norte, S = coches dirección sur.

Todas las variables anteriores inician su valor en 0. A continuación, el orden de ejecución de las funciones del monitor en el que se modifican estas variables viene en las funciones car ó pedestrian.

Para la variable P, la función pedestrian nos indica que primero se ejecuta la función can_enter_pedestrian() del monitor, donde el valor de P aumenta 1 unidad y posteriormente leaves_pedestrian(), donde el valor de P disminuye 1. Como el orden NO se puede invertir, $P \geq 0$.

Análogamente obtenemos $N \geq 0 \wedge S \geq 0$ con la función car y las funciones wants_enter_car(direction) y leaves_car(direction).

Por otro lado, miremos cada función del monitor y veamos que su invariante es:

$$\left.\begin{array}{l}(P \geq 0 \wedge N \geq 0 \wedge S \geq 0) \wedge \\ (P > 0 \longrightarrow N = 0 \wedge S = 0) \wedge (N > 0 \longrightarrow P = 0 \wedge S = 0) \wedge (S > 0 \longrightarrow P = 0 \wedge N = 0)\end{array}\right\} I$$

- wants_enter_car(direction):

Sea el caso dirección = 0, ya hemos visto $P \geq 0$, $N \geq 0$, $S \geq 0$.

La primera condición es que $P = 0 \wedge S = 0$, luego la única varible que puede ser mayor que 0 es N. Además la función sólo puede aumentar el valor de N, luego I se verifica.

Análogamente el caso dirección = 1.

Análogamente se comprueba la función wants-enter-pedestrian().

- leaves-car (direction)

Sea el caso dirección = 0, sabemos que $N > 0$, $P = 0$, $S = 0$ (porque primero ejecutamos wants-enter-car(0)).

En la función se reduce el valor de N en una unidad, luego obtenemos $N \geq 0$. Además si $N = 0$, mandamos señal a la condición can-cars-1, primero, en la función can-enter-car(1) donde se verifica I ó a la condición can-pedestrians en la función can-enter-pedestrian donde igualmente se verifica.

Análogamente se comprueba el caso diretion = 1.

Análogamente se comprueba leaves-pedestrian. □

- Vamos a ver que ∄ deadlock.

Supongamos que todos los procesos están esperando, esto es:

i) - pi, proceso pedestrian, no cumple la condición can-pedestrians, luego $N>0$ v $S>0$ (el caso $N>0 \wedge S>0$ NO se puede dar por el invariante del monitor)

ii) - P_j, proceso car(0), no cumple la condición can_cars_0, luego $S>0$ v $P>0$ (el caso $S>0 \wedge P>0$ NO se puede dar por el invariante del monitor)

iii) - P_k, proceso car(1), no cumple la condición can_cars_1, luego $N>0$ v $P>0$ (el caso $N>0 \wedge P>0$ NO se puede dar por el invariante del monitor)

Veamos caso por caso que eso no puede ocurrir:

Si en (i) tenemos que $N>0$, por el invariante: $S=0 \wedge P=0$, luego (ii) no se cumple, no podría estar esperando el proceso P_j.

Si en (i) tenemos que $S>0$, por el invariante: $N=0 \wedge P=0$, luego (iii) no se cumple, no podría estar esperando el proceso P_k.

Si en (ii) tenemos que $S>0$, por el invariante: $N=0 \wedge P=0$, luego (iii) no se cumple, no podría estar esperando el proceso P_k.

Si en (iii) tenemos que $P>0$, por el invariante: $N=0 \wedge S=0$, luego (i) no se cumple, no podría estar esperando el proceso P_i.

Si en (iii) tenemos que $N > 0$, por el invariante: $P = 0 \wedge S = 0$, luego no se cumple, no podría estar esperando el proceso P_i.

Si en (iii) tenemos que $P > 0$, por el invariante: $N = 0 \wedge S = 0$, luego (i) no se cumple, no podría estar esperando el proceso P_i.

Como ningún caso posible puede ocurrir, al menos un proceso no estará en espera. □

- Sin embargo, en esta solución sí hay inanición: imaginemos que hay un coche que va hacia el norte en el puente, entonces llega otro antes de que el primero se salga y así sucesivamente. Los procesos que pertenezcan al grupo de coches que vayan hacia el sur ó de peatones se quedarán esperando eternamente sin poder pasar (cuando hablamos de cardinales infinitos).

Para evitar la inanición, vamos a crear 4 nuevas variables en el monitor de forma que una de ellas sea la variable turno y que vaya rotando por los distintos grupos. Así solucionamos el problema anterior, pero debemos tener en cuenta que si no hay coches en la cola de uno de los grupos, se crearía un deadlock, pues todos los procesos quedarían esperando. Para evitar lo anterior, las 3 variables nuevas restantes son las que indican el número de procesos en cola de cada grupo. Es necesario entonces que modifiquemos las condiciones a cumplir de las variables condiciones. Así la solución mejorada quedaría:

```
monitor Puente
    integer ncars_0 = 0
    integer ncars_1 = 0
    integer npedestrians = 0
    integer ncarsWaiting_0 = 0
    integer ncarsWaiting_1 = 0
    integer npedWaiting = 0
    integer turn = 0    # turn ∈ {0 (N), 1 (S), 2 (P)}
    condition can_cars_0, can_cars_1, can_pedestrians

    operation canEnter_carsN():
        return ((npedestrian == 0 and (turn != 2 or npedWaiting == 0)) and
               (ncars_1 == 0 and (turn != 1 or ncarsWaiting_1 == 0)))
```

```
operation canEnter_carsS():
  return ((npedestrian == 0 and (turn != 2 or npedWaiting == 0)) and
         (ncars_0 == 0 and (turn != 0 or ncarsWaiting_0 == 0)))

operation canEnter_pedestrians():
  return ((ncars_0 == 0 and (turn != 0 or ncarsWaiting_0 == 0)) and
         (ncars_1 == 0 and (turn != 1 or ncarsWaiting_1 == 0)))


operation wants_enter_car(direction)   # direction==0 (Norte) , direction == 1 (sur)
        if direction == 0 :
            ncarsWaiting_0 += 1
            can_cars_0.waitC(canEnter_carsN)
            carsWaiting_0 -= 1
            ncars_0 += 1
        else :  # direction == 1
            ncarsWaiting_1 += 1
            can_cars_1.waitC(canEnter_carsS)
            ncarsWaiting_1 -= 1
```

```
        ncars_1  += 1



operation  Leaves_car (direction):
    if  direction == 0:
        ncars_0  -= 1
        if  ncarsWaiting_1  > 0
            turn = 1
        elif  npedWaiting  > 0
            turn = 2
        if  ncars_0 == 0.
            signalC (can_cars_1)
            signalC (can_pedestrians)
    else:
        ncars_1 -= 1
        if  npedWaiting > 0
            turn = 2
        elif ncarsWaiting_0  > 0
            turn = 0
        if  n_cars_1  == 0:
```

```
        signalC(can-pedestrians)
        signalC(can-cars-0)


operation wants-enter-pedestrian():
    npedWaiting += 1
    can-pedestrians.waitC(canEnter-pedestrians)
    npedWaiting -= 1
    npedestrians += 1


operation leaves-pedestrian():
    npedestrians -= 1
    if ncarsWaiting-0 > 0
        turn = 0
    elif ncarsWaiting-1 > 0
        turn = 1
    if npedestrians == 0:
        signalC(ncars-0)
        signalC(ncars-1)
```

- Sea $WN = ncarWaiting\_0$, $WS = ncarsWaiting-1$, $WP = npedWaiting$, $T = turn$

EL invariante del monitor es:

$I \wedge (WN \geq 0 \wedge WS \geq 0 \wedge WP \geq 0 \wedge 0 \leq T \leq 2)$

- Ausencia de deadlock: (No puede pasar nadie al puente, // éste quedará vacío)

Supongamos la solución podría llegar al deadlock, entonces por las las condiciones i) canEnter_carsN, ii) canEnter_carsS y iii) canEnter_pedestrians, tendríamos:

por (i): $P>0 \vee (T=2 \wedge WP>0) \vee S>0 \vee (T=1 \wedge WS>0)$

por (ii): $P>0 \vee (T=2 \wedge WP>0) \vee N>0 \vee (T=0 \wedge WN>0)$

por (iii): $N>0 \vee (T=0 \wedge WN>0) \vee S>0 \vee (T=1 \wedge WS>0)$

Vamos a distinguir los siguientes 3 casos:

- Sea $T=0$ entonces tendríamos:

por (i): $P>0 \vee S>0$ → No puede haber nadie en el puente ⊛

por (ii): $P>0 \vee N>0 \vee WN>0$

por (iii): $N>0 \vee WN>0 \vee S>0$ →

⊛ Si $P>0$ (como por el invariante $N=0$ y $S=0$) entonces $WN>0$

Pero sabemos que finalmente P será $P=0$ $\Rightarrow$ canEnter_carsN = True

y al tener $P=0$ se notifica a la condición can_cars_0.

Análogamente pasa con $S>0$

Como (i) no puede cumplirse, $\nexists$ deadlock para el caso $T=0$.

- $T = 1$:

  por (i) : $P>0 \lor S>0 \lor (T=1 \land WS>0)$

  por (ii) : $P>0 \lor N>0$ $\longrightarrow$ No se puede dar (análogo $T=0$)

  por (iii) : $N>0 \lor S>0 \lor (T=1 \land WS>0)$

- $T = 2$:

  por (i) : $P>0 \lor (T=2 \land WP>0) \lor S>0$

  por (ii) : $P>0 \lor (T=2 \land WP>0) \lor N>0$

  por (iii) $N>0 \lor S>0$ $\longrightarrow$ No se puede dar (análogo $T=0$)

Luego como no hay deadlock para ningún valor posible de la luego como no hay deadlock para ningún valor posible de la variable T (turn), hay ausencia de deadlock. □

- Finalmente, veamos que no hay inanición. En este algoritmo nos aseguramos que una vez que salga alguien del puente, exista la posibilidad de cambiar el turno si procesos de otro grupo están esperando.

Así consideremos, por ejemplo, que hay coches hacia el Norte ($N>0$) en el puente, cuando salga un coche del puente tenemos las condiciones:

1º) si $ws>0 \longrightarrow T=1$ (el turno cambia a los coches del sur)

si lo anterior no se cumple,

2º) si $WP>0 \longrightarrow T=2$ (el turno cambia a los peatones)

Cuando deje de haber coches ($N=0$) en el puente, mandará una señal a las condiciones can_cars_0 y can_pedestrians respectivamente, luego:

si $ws>0$ ($T=1$, $\overbrace{N=0 \text{ y } P=0}^{\text{pues no hay nadie en el puente}}$), podrán pasar los que van hacia el sur a continuación.

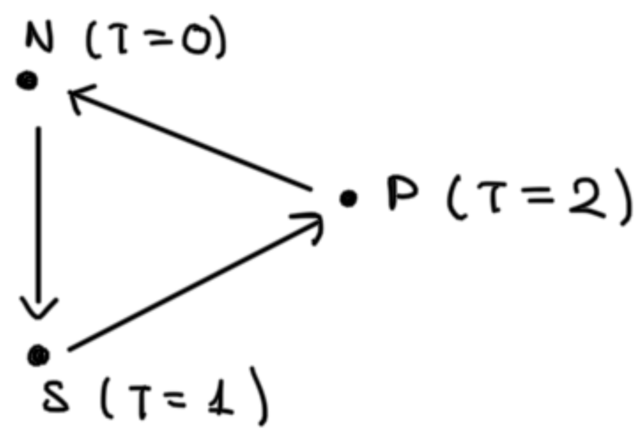
si $Ws=0$ y $WP>0$ ($T=2$, $N=0$ y $S=0$), podrán pasar los peatones

si $Ws=0$ y $WP=0$ ($T=0$, $S=0$ y $P=0$), podrán pasar los que van hacia el norte.

Ahora bien, sin pérdida de generalidad, imaginemos que ha pasado por el puente una tanda de coches que van hacia el norte y posteriormente una tanda de coches que van hacia el sur. Veremos que si hay peatones esperando ($WP > 0$), el turno pasa a ser de los peatones:

Cuando sale el primer coche de la tanda de coches hacia el sur, como en leaves_car(1) tenemos la condición (primera) si $WP > 0 \longrightarrow T = 2$, el turno pasa a los peatones. Cuando no haya más coches que vayan hacia el sur en el puente, se manda señal a la condición can_pedestrians y como: $T = 2$, $N = 0$ y $S = 0$, los peatones pueden pasar.

El orden de la solución sería:



Luego funcionaría para todos los casos por el orden que hemos implementado las condiciones para cambiar el valor del turno y el orden en dar las notificaciones a las variables condición. □