

Solución a papel (Práctica 2)

Fernán González Pereira

Vamos a comenzar exponiendo una solución sencilla y la vamos a mejorar para llegar a una que cumpla

- El puente es seguro
- Ausencia de deadlocks
- Ausencia de inanición

la cual hemos implementado en python.

Monitor

South-car: int = 0

North-car: int = 0

Pedestrians: int = 0

South-car-condition: VC

North-car-condition: VC

Pedestrian-condition: VC

• Invariant: $\{ \text{South-car} \geq 0 \wedge \text{North-car} \geq 0 \wedge \text{Pedestrians} \geq 0$
 $\text{South-car} > 0 \rightarrow \text{North-car} = 0 \wedge \text{Pedestrians} = 0$
 $\text{North-car} > 0 \rightarrow \text{South-car} = 0 \wedge \text{Pedestrians} = 0$
 $\text{Pedestrians} > 0 \rightarrow \text{North-car} = 0 \wedge \text{South-car} = 0 \}$

wants-enter-car (direction)

if direction == South

South-car-condition.wait($\text{North-car} == 0 \wedge \text{Pedestrians} == 0$)

South-car += 1

else

North-car-condition.wait($\text{South-car} == 0 \wedge \text{Pedestrians} == 0$)

North-car += 1

leaves-car (direction)

if direction == South

{Inv \wedge South-car > 0}

South-car -= 1

if South-car == 0

North-car-condition.notify()

Pedestrian-condition.notify()

else {Inv \wedge North-car > 0}

if North-car == 0

South-car-condition.notify()

Pedestrian-condition.notify()

wants_enter_pedestrian():

pedestrian_condition.wait(north_car == 0 & south_car == 0)

pedestrians += 1

leaves_pedestrian():

{Inv 1 pedestrians > 0}

pedestrians -= 1

if pedestrians == 0:

north_car_condition.notify()

south_car_condition.notify()

• car(direction)

loop

monitor.wants_enter_car(direction)

operaciones cruzar puente

monitor.leaves_car(direction)

• pedestrian()

loop

monitor.wants_enter_pedestrian()

operaciones cruzar puente

monitor.leaves_pedestrian()

Esta solución cumple que el puente es seguro ya que para que un coche pueda cruzar el puente no puede haber coches en dirección contraria y tampoco peatones. Para un peatón se debe cumplir que no haya coches en ninguno de los dos sentidos. Por tanto, la circulación es segura. Esto se consigue con los wait for en wants_enter_car() y wants_enter_pedestrian()

En esta solución encontramos problemas de inanición. Puede darse el caso en que hay un proceso (coche en cualquier sentido, o peatón) que se quede esperando indefinidamente. Supongamos que un peatón quiere cruzar, el cual queda bloqueado hasta que el puente este vacío. Si pasa un coche en un sentido poder seguir pasando coches en un mismo sentido y nunca se notifica que el puente queda liberado, por lo que el peatón se queda indefinidamente esperando.

Para solucionar esto vamos a introducir una variable que indique cuantos están esperando, de cada tipo, para acceder al puerto y un sistema de turnos para que todos accedan al puerto, y así solucionar la inanición.

Monitor:

South-car: int = 0

North-car: int = 0

pedestrians: int = 0

South-car-waiting: int = 0

North-car-waiting: int = 0

pedestrian-waiting: int = 0

South-car-condition: VC

North-car-condition: VC

pedestrian-condition: VC

turn: int = North

los turnos son North = 0; South = 1; Ped = 2

• Inu: { South-car ≥ 0 \wedge North-car ≥ 0 \wedge pedestrians ≥ 0

North-car-waiting ≥ 0 \wedge South-car-waiting ≥ 0 \wedge pedestrian-waiting ≥ 0

South-car $> 0 \rightarrow$ North-car = 0 \wedge pedestrians = 0

North-car $> 0 \rightarrow$ South-car = 0 \wedge pedestrians = 0

pedestrians $> 0 \rightarrow$ South-car = 0 \wedge North-car = 0 }

wants-enter-car(direction)

if direction == South

South-car-waiting += 1

{Inu \wedge South-car-waiting > 0 }

South-car-condition.wait((North-car == 0 \wedge pedestrians == 0) \wedge ((North-car-waiting ≤ 5 \wedge pedestrians-waiting ≤ 5) \vee turn == South))

South-car-waiting -= 1

South-car += 1

else:

North-car-waiting += 1

{Inu \wedge North-car-waiting > 0 }

North-car-condition.wait((South-car == 0 \wedge pedestrians == 0) \wedge ((South-car-waiting ≤ 5 \wedge pedestrians-waiting ≤ 5) \vee turn == North))

North-car-waiting -= 1

North-car += 1

leaves-car (Direction):

if direction == South:

{ Inv 1 south-car > 0 }

south-car -= 1

if pedestrians_waiting > 0:

turn = ped

else if north-car_waiting > 0:

turn = North

if south-car == 0:

north-car-condition.notify()

pedestrian-condition.notify()

else:

{ Inv 1 north-car > 0 }

north-car -= 1

if south-car_waiting > 0:

turn = south

else if pedestrians_waiting > 0:

turn = ped

if north-car == 0:

south-car-condition.notify()

pedestrian-condition.notify()

wants-enter-pedestrian()

pedestrians_waiting += 1

{ Inv 1 pedestrian_waiting > 0 }

pedestrian-condition.wait((north-car == 0 & south-car == 0) & ((north-car_waiting < 5
& south-car_waiting < 5) || turn == ped))

pedestrian_waiting -= 1

pedestrians += 1

leaves - pedestrian(1):

{Inu n pedestrians > 0}

pedestrians -= 1

if north-car-waiting > 0:

turn = North

else if south-car-waiting > 0:

turn = south

if pedestrians == 0:

south-car-condition.notify()

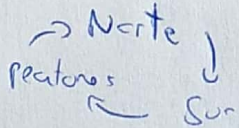
north-car-condition.notify()

Esta solución sigue cumpliendo la seguridad en la circulación.

• Ausencia de deadlocks: Si dos procesos se impiden mutuamente cruzar el puente estaríamos ante un caso de deadlock y el programa no avanza. Esto no ocurre ya que a un proceso le cabe que le impide cruzar es que haya proceso cruzando. Si el proceso que quiere cruzar, como tiene gente esperando, antes o después le llegará el momento de cruzar cuando el puente se vacíe.

Si a la hora de ceder los turnos no metemos la condición de que se ceda si el proceso tiene gente esperando le estaríamos cediendo el turno a un proceso sin nadie esperando. Los otros dos procesos pueden tener gente esperando y que este número sea superior a 5. Como no es el turno de ninguno de los 2 podrían cruzar y en los otros dos procesos no supera a 5 la gente esperando, pero esto no ocurre luego estos 2 procesos se bloquean y el programa no avanza, teniendo así un caso de deadlock.

• Ausencia de inanición hemos establecido el siguiente sistema de rotación de turnos. Si están cruzando coches del norte, al salir, se le cede el turno a los del sur - si tienen coches esperando, y si esto no ocurre se le cede a los peatones si tienen gente esperando. Esta misma idea la aplicamos en el resto de casos con el siguiente orden:



```
graph TD; Norte --> Peatones; Peatones --> Sur;
```

De esta forma estamos estableciendo un sistema rotatorio de turnos que hace que todos los procesos antes o después vayan a cruzar si tienen gente esperando, pues si el puente está vacío y es su turno el proceso puede cruzar. Puede darse el caso en que un proceso cruce mientras no es su turno y para garantizar que no se acumule gente esperando en los otros procesos, imponemos como condición que han de estar esperando menos o 5 personas en los otros procesos. Tras estos cambios resolvemos el problema de inanición.