Facultad de Ciencias Matemáticas

Programación Paralela

curso 2021-2022

Práctica 2: Túnel de Kiyotaki

Esther Arribas García, Lucía Bragado Pérez y Álvaro Seco Silva

Abril, 2022

Versión básica

En esta primera versión se permite entrar a un coche en el túnel si el túnel está vacío y no hay coches dentro del túnel en sentido opuesto. Así, si empiezan a entrar coches en dirección norte, los que se encuentran en el sur deberán esperar a que todos ellos salgan del túnel. Para llevar a cabo esta idea hemos simulado el monitor a través de semáforos en Python, junto con determinadas variables condición, que permiten la entrada o no al túnel.

Esta idea para resolver la práctica es la más sencilla posible, pero al mismo tiempo da lugar a los siguientes problemas: ¿Qué pasaría si tenemos una grandísima cantidad de coches en sentido norte, y estos comienzan a entrar en el túnel? ¿Podrían después los de sentido sur entrar, independientemente de cuántos coches haya en sentido norte queriendo entrar en el túnel? Nos encontramos ante un problema de posible inanición, pues, en este caso, los coches en sentido sur deben esperar a que salgan los del norte, pero si son muchísimos podría ocurrir que tardasen demasiado o incluso que nunca saliesen todos (si hay infinitos coches en ese sentido). De igual manera ocurriría en el otro sentido. Es por esta razón que presentamos las tres versiones siguientes, basadas en establecer turnos por tiempo o cantidad de coches con el fin de dar respuesta al problema de inanición.

Versión tiempo

En esta solución usaremos monitores (mejor dicho, los simularemos) junto con variables condición creando turnos por tiempo entre los sentidos. Los turnos están controlados por un tiempo fijo t que representa el tiempo máximo en el túnel para los coches de un sentido. Se cambiará de turno cuando se haya finalizado el tiempo t y se compruebe que el túnel está vacío (es decir, que no haya ningún coche dentro del túnel en el sentido opuesto). Con este sistema se consigue que cada cierto tiempo t se cambie de sentido y así evitar que uno de los dos sentidos se quede una eternidad esperando a poder entrar al túnel.

Versión tiempo 2

Esta solución es igual a la versión tiempo salvo en el hecho de que si es el turno de un sentido pero no hay coches esperando a entrar en ese sentido, entonces se cambia el turno al sentido opuesto. Los turnos seguirán controlados por un tiempo fijo t, se cambiará de turno cuando se haya finalizado el tiempo t y se compruebe que el túnel está vacío. Si en un sentido no hay coches esperando a entrar, se cambia el turno, sin tener que esperar el tiempo t. Así, damos solución a la situación que puede ocurrir en la versión tiempo si se tiene que esperar un tiempo t para que no pase ningún coche.

Versión contador de coches

En esta solución usaremos monitores, junto con el uso de variables condición, los turnos y añadiremos el número de coches esperando en un sentido y el otro. La idea de esta versión consiste en establecer un número máximo de coches por turno x. Se cambiará de turno cuando hayan finalizado de pasar el túnel los x coches que han entrado y se compruebe que el túnel este vacío. Además, añadiremos la condición de que, si en un lado hay menos de x coches esperando a entrar, estos coches pasen y se cambie de turno, sin tener que esperar a que pasen los x coches en dicho sentido (pues podrían no existir). De esta forma, los turnos determinarán para qué sentido está abierto el túnel y qué coches pueden entrar en el mismo. Con esta solución se consigue que cada x coches que pasen se cambie de sentido y así evitar que uno de los dos sentidos se quede mucho tiempo esperando a poder entrar al túnel, resolviendo el problema de inanición.

EXTRA

Se adjunta un documento con la demostración del invariante que se satisface en el programa en la versión básica y en la versión tiempo, junto con un diagrama de árbol auxiliar de la solución básica.

INVARIANTE (inside north ∈ ℕ, inside north ≥ 0 ; inside south ∈ ℕ, inside south ≥ 0)

$$\text{inside north} > 0 \rightarrow \text{inside south} = 0$$

$$\text{inside south} > 0 \rightarrow \text{inside north} = 0$$

```
wants_enter(direction)
 {INV}
 si direction == NORTH:
     empty_south.wait(south_inside==0)
     {INV ∧ SOUTH_INSIDE=0}
     north_inside+=1
     {INV}
 si direction == SOUTH:
     empty_north.wait(north_inside==0)
     {INV ∧ NORTH_INSIDE=0}
     south_inside+=1
     {INV}
leaves_tunnel(direction)
{INV}
 si direction == NORTH:
     north_inside -= 1
     si north_inside==0:
         {INV ∧ NORTH_INSIDE=0}
         empty_north.signal()
si direction == SOUTH:
    south_inside -= 1
    si south_inside==0:
        {INV ∧ SOUTH_INSIDE=0}
        empty_south.signal()
{INV}
```

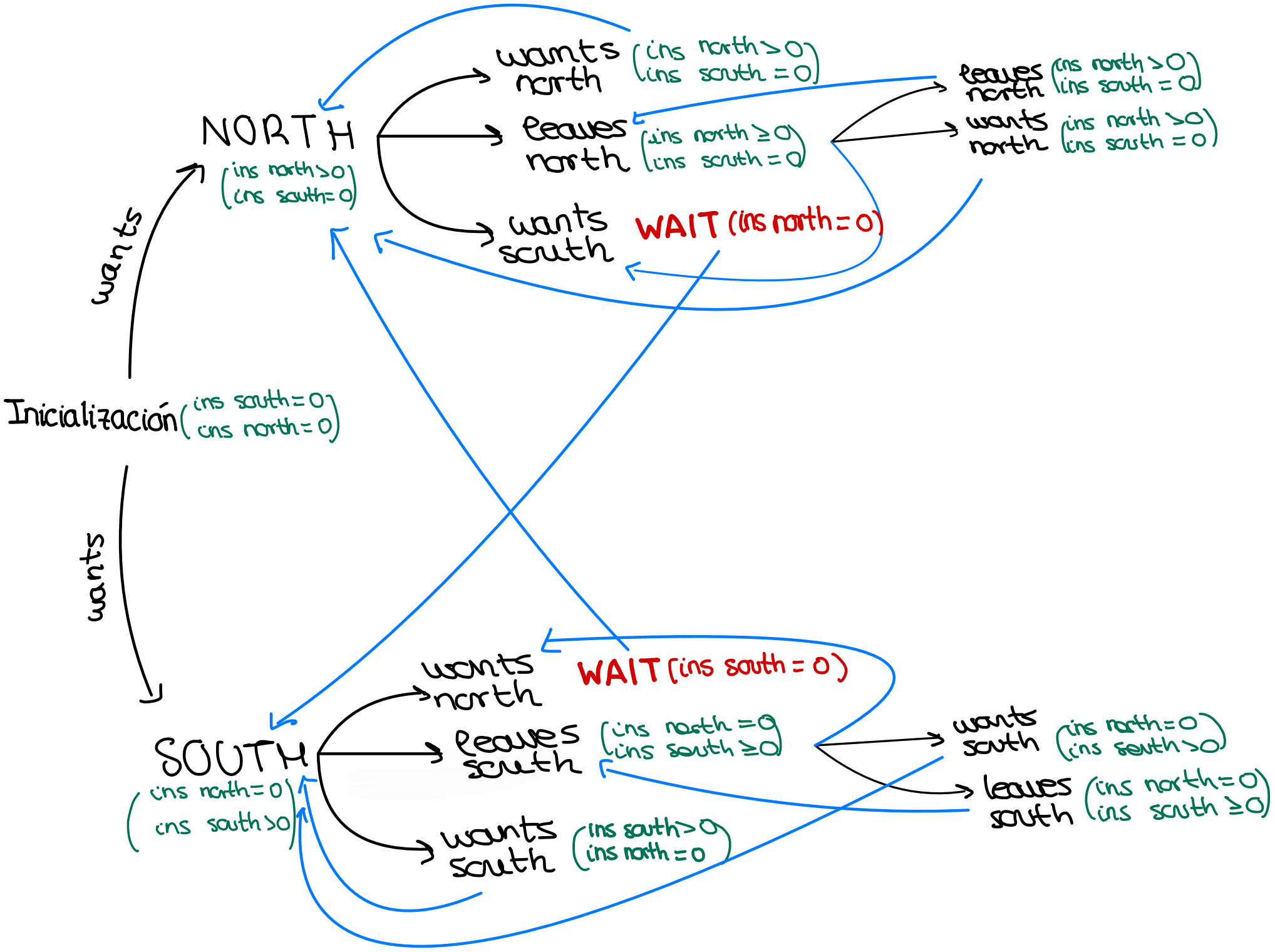
INVARIANTE

(versión tiempo)

(inside north ∈ ℕ, inside north ≥ 0
inside south ∈ ℕ, inside south ≥ 0)

> inside north > 0 → inside south = 0
>
> inside south > 0 → inside north = 0

```
wants_enter(direction)
 {INV}
  si direction == NORTH:
     turno.wait(turno == north)
     {INV ∧ turn=0 ∧ SOUTH_INSIDE=0}
     si tiempo == 0:
         tiempo = tiempo_transcurrido
      north_inside += 1
     {INV}
  si direction == SOUTH:
     turno.wait(turno == SUR)
     {INV ∧ turn=1 ∧ NORTH_INSIDE=0}
      si tiempo == 0:
          tiempo = tiempo_transcurrido
       south_inside += 1
      {INV}
 leaves_tunnel(direction)
 {INV}
 si direction == NORTH:
       north_inside -= 1
       si tiempo - tiempo_transcurrido > T
            turno = 1
            tiempo = 0
            turno.signal()
   si direction == SOUTH:
       south_inside -= 1
       si tiempo - tiempo_transcurrido > T:
            turno = 0
            tiempo = 0
            turno.signal()
 {INV}
```


Inicialización
(ins south = 0
ins north = 0)
wants
NORTH
(ins north > 0
ins south = 0)
wants north
(ins north > 0
ins south = 0)
leaves north
(ins north ≥ 0
ins south = 0)
wants south
leaves north
(ins north > 0
ins south = 0)
wants north
(ins north > 0
ins south = 0)
WAIT (ins north = 0)
wants
SOUTH
(ins north = 0
ins south > 0)
wants north
leaves south
(ins north = 0
ins south ≥ 0)
wants south
(ins south > 0
ins north = 0)
WAIT (ins south = 0)
wants south
(ins north = 0
ins south > 0)
leaves south
(ins north = 0
ins south ≥ 0)