Vehiculo: *Se inicializa con el tipo de vehículo que es.*

Por generalidad, al funcionar todos los que pasan por el túnel de la misma forma los agrupamos bajo la clase Vehiculo de forma que tendríamos 3 tipos: Coche en dirección norte, coche en dirección sur y peatón (Usamos una enumeración: Tipos que los contiene para manejarlos).

Entrar en túnel():

self.monitor.esperaEntrar(tipo)

{El proceso correspondiente a lo que hace en el túnel. (Se ha implementado como un delay de duración condicionada por el tipo.)}

self.monitor.sale(tipo)

Monitor:

Inicialización:

Invariantes:

- *Para un tipo t de Tipos, si num[t]>0 => para todo tipo nt de Tipos\{t}, tenemos num[nt]=0 y cond[nt]=False*
- *cond[t] = True <=> para todo tipo nt de Tipos\{t}, tenemos num[nt]=0*
- *num[t]>=0 para todo tipo t de Tipos*

cond = {tipo: cv = False for tipo in Tipos}

num = {tipo: 0 for tipo in Tipos}

lock = Lock

esperaEntrar (t):

Se presupone que el invariante se cumple al llamar a esta función

lock.acquire()

cond[t].wait([num[nt]=0 para todo nt en Tipos\{t})

Si un tipo está esperando aquí significa que alguno de los otros está dentro del túnel, por lo que tarde o temprano, saldrá y este tendrá la oportunidad de entrar. Cuando se hace cierto para un tipo, se puede hacer para los de otros tipos también, pero bajo la hipótesis de justicia, como esto se hace cierto un número indefinido de veces, eventualmente todos entrarán.

Se presenta un posible problema de inanición si cada vez que salen los de un tipo vuelven a entrar los de ese mismo, o si siguen entrando hasta pasar todos. Haciendo que el resto esperen a que hayan terminado sus procesos para entrar.

num[t] ++

Como num[t] seguro que no es 0 ahora, las condiciones del resto de tipos ya no se cumplen

Hemos confirmado que el valor de num del resto de tipos fuera 0 antes de entrar, no han podido modificarlo porque tenemos cogido el lock del túnel y tampoco podrán hasta que sus

condiciones vuelvan a ser ciertas, lo que requiere que num[t]=0, por lo que se mantendrá el invariante.

lock.release()

Al salir de la función tenemos garantizado, además del invariante, que:

num[nt] = 0 para nt en Tipos\{t}

num[t] > 0

sale(t):

Se presupone que se cumple el invariante al llamar a esta función

Además, al llamarla después de un esperarEntrar(t), también se presupone que num[nt] = 0 para nt en Tipos\{t} y num[t] > 0

lock.acquire()

num[t] --

Como al entrar num[t] > 0 se sigue cumpliendo que num[t] >= 0

cond[nt].notify_all() para todo nt en Tipos\{t}

lock.release()

Monitor con posible solución para el problema de inanición:

Inicialización: *(Se añaden las variables numEspera, tiempoActual y ultEnPasar. A parte de la constante que se presupone que tiene predefinida de tMax y la función para ver el tiempoActual (en Python time.time()).)*

Invariantes:

- *Para un tipo t de Tipos, si num[t]>0 => ultEnPasar[t]=1 y para todo tipo nt de Tipos\{t} cond[nt]=False*
- *cond[t] = True <=> para todo tipo nt de Tipos\{t}, tenemos num[nt]=0 y pasa almenos una de las siguientes 3 cosas:*
  - *numEspera[t]=0 para todo t*
  - *tiempoActual() – tiempoT < tMax*
  - *ultEnPasar[t]=0*
- *num[t]>=0 para todo tipo t de Tipos*
- *numEspera[t]>=0 para todo tipo t de Tipos*
- *tiempoT >= 0*
- *ultEnPasar[t]∈{0,1} para todo tipo t de Tipos*
- *solo hay como mucho un t para el que ultEnPasar[t]=1 en cada momento*
- *ultEnPasar[t]=1 => para todo tipo nt de Tipos\{t}, tenemos num[nt]=0 y ultEnPasar[nt]=0*

```
cond = {tipo: False    for tipo in Tipos}

num  = {tipo: 0        for tipo in Tipos}

lock = Lock

numEspera  = {tipo: 0      for tipo in Tipos}

tiempoT  = 0

ultEnPasar  = {tipo: 0      for tipo in Tipos}
```

esperaEntrar (t): *(Se modifica la condición, y se añaden las instrucciones necesarios para mantener los invariantes y que funcione el control del tiempo y número en espera)*

Se presupone que el invariante se cumple al llamar a esta función

```
lock.acquire()

cond[t].wait(([num[nt]=0 para todo nt en Tipos\{t}) ∧ ((ultEnPasar[t]= 0) ∨
(tiempoActual()-tiempoT) < tMax) ∨ (numEspera[nt]=0 para todo nt en Tipos\{t})))
```

Si un tipo está esperando aquí significa que 1 de 3 posibles situaciones ocurre:

- Hay uno de otro tipo dentro del túnel, por lo que tarde o temprano, saldrá y este tendrá la oportunidad de entrar.

- Acaba de salir está dentro del túnel uno de este tipo, ha pasado más tiempo que TMax desde que entró el primero de este tipo y hay de otro tipo esperando a entrar. Una vez haya pasado el que está en espera ultEnPasar[t]= 0 por lo que la condición se cumplirá y podrá entrar este.

Cuando se hace cierto para los de este tipo, también podría hacerse cierto para los de otro tipo, pero bajo la hipótesis de justicia, como esto se hace cierto un número indefinido de veces, eventualmente todos entrarán.

Se presenta un posible problema de inanición si cada vez que salen los de un tipo vuelven a entrar los de ese mismo, o si siguen entrando hasta pasar todos. Haciendo que el resto esperen a que hayan terminado sus procesos para entrar.

numEspera[t] --

Como antes del wait habíamos incrementado numEspera[t], este era mayor que 0, por lo que ahora es mayor o igual que 0, cumpliendo el invariante

SI (ultEnPasar[t]) == 0 ∨ (self.numEspera[nt] = 0 para todo nt en Tipos\{t}) ENTONCES

tiempoT = tiempoActual

ultEnPasar[t] = (t == tipo) para todo tipo en Tipos

FINSI

Se cambia el valor de ultEnPasar para todos los Tipos para que verifique el invariante y se reactualiza el tiempo para controlar cuanto tiempo ha pasado desde que entró el primero.

num[t] ++

Como num[t] seguro que no es 0 ahora, las condiciones del resto de tipos ya no se cumplen

Hemos confirmado que el valor de num del resto de tipos fuera 0 antes de entrar, no han podido modificarlo porque tenemos cogido el lock del túnel y tampoco podrán hasta que sus condiciones vuelvan a ser ciertas, lo que requiere que num[t]=0, por lo que se mantendrá el invariante. Algo similar pasa con ultEnPasar.

lock.release()

Al salir de la función tenemos garantizado, además del invariante, que:

num[nt] = 0 para nt en Tipos\{t}

num[t] > 0

utlEnPasar[t] = 1

sale(t): *(Se mantiene igual que antes)*

Se presupone que se cumple el invariante al llamar a esta función

Además, al llamarla después de un esperarEntrar(t), también se presupone que num[nt] = 0 para nt en Tipos\{t} y num[t] > 0

lock.acquire()

num[t] -= 1

Como al entrar num[t] > 0 se sigue cumpliendo que num[t] >= 0

cond[nt].notify_all() para todo nt en Tipos\{t}

lock.release()