Invariante y funcionamiento

Lucía del Nido Herranz, Victor Güejes Cepeda, Celia Vaquero Espinosa.

Primera solución:

El funcionamiento es: si hay coches pasando en una dirección, los del sentido contrario no podrán pasar hasta que no pasen todos los de la otra dirección. Una vez ya no queden coches que quieran pasar en ese sentido, entrarán los del sentido contrario que estuviesen esperando. Esta solución puede causar inanición si hay suficientes coches pasando en un único sentido.

COCHE (proceso)

{ INVARIANTE:
1. inside_north >= 0
2. inside_south >= 0
3. inside_north > 0 => inside_south = 0
4. inside_south > 0 => inside_north = 0
}

wants_enter()

leaves_tunel()

Variables del invariante:

- inside_north: la cantidad de coches dirección norte que hay dentro del túnel dirección norte.

(para las variables south es análogo)

Condiciones 1 y 2: Condiciones de existencia, todas las variables tienen que ser enteras no negativas.

Condiciones 3 y 4: Si hay coches pasando en un sentido, no puede haber coches pasando en el contrario.

Vamos a llevar a cabo la demostración del invariante en la segunda solución ya que el invariante de la primera está contenido en el de la segunda.

Solución segunda:

Para evitar la inanición, si hay coches pasando en un sentido y hay algún coche esperando en sentido contrario, hay un máximo de coches MAX que pueden pasar. Si no hay coches esperando en sentido contrario, los coches pueden seguir pasando. Esta solución tiene problemas cuando la distribución de coches es asimétrica, ya que es una solución injusta para la fila de coches más larga.

COCHE (proceso)

{ INVARIANTE:
1. inside_north >= 0
2. inside_south >= 0
3. waiting_north >= 0
4. waiting_south >= 0
5. contador_south >= 0
6. contador_north >= 0
7. inside_north > 0 => inside_south = 0
8. inside_south > 0 => inside_north = 0
9. waiting_north > 0 => contador_south < MAX
10. waiting_south > 0 => contador_north < MAX
}

wants_enter()

leaves_tunel()

Variables del invariante:

- inside_north: la cantidad de coches dirección norte que hay dentro del túnel dirección norte.
- waiting_north (añadida): la cantidad de coches dirección norte que están esperando para entrar al túnel.
- counter_north (añadida): la cantidad de coches dirección norte que han entrado al túnel desde el último cambio de sentido.

(para las variables south es análogo)

Condiciones 1-6: Condiciones de existencia, todas las variables tienen que ser enteras no negativas.

Condiciones 7 y 8: Si hay coches pasando en un sentido, no puede haber coches pasando en el contrario.

Condiciones 9 y 10 (añadidas): Si está esperando algún coche en sentido contrario, sólo pueden pasar una cantidad máxima MAX de coches y se cambia la dirección en la que pasan. Esto evita la inanición.

Durante todo el proceso de cada coche, las condiciones de existencia (1 a 6) y las condiciones 7 y 8, se van a verificar siempre. Que las variables 9 y 10 sean invariantes al inicio y al final del proceso viene del uso del monitor.

WANTS_ENTER()

```
def wants_enter(self, direction):
```

{Aquí se cumple todo el invariante}

```
        self.mutex.acquire()
        if direction == SOUTH:
                self.waiting_south.value = self.waiting_south.value + 1
```

(4) waiting_south >= 0 se sigue verificando si le sumamos 1, y queda waiting_south>0

```
                print(f"waiting_south: {self.waiting_south.value}")
                self.sem_north.wait_for(self.no_cars_north_inside)
```

Cuando el monitor se abre, se verifica que:

inside_north.value = 0 y (contador_south.value < MAX o waiting_north.value ==0)

Por tanto, el coche que hasta ahora estaba esperando para poder pasar, ya puede meterse en el túnel.

```
                self.inside_south.value = self.inside_south.value + 1
```

Podemos aumentar inside_south ya que como inside_north.value = 0, se va a seguir verificando (8) inside_south > 0 => inside_north = 0 y (2) inside_south >= 0, que es algo que queremos que ocurra en todo momento.

```
                self.contador_south.value = self.contador_south.value + 1
```

(5) contador_south >= 0 se sigue verificando si aumentamos la variable.

```
                print(f"contador_south: {self.contador_south.value}")
                self.waiting_south.value = self.waiting_south.value - 1
```

En este punto waiting_south era estrictamente positivo, por lo que al restarle 1 se sigue cumpliendo (4).

Por tanto, al salir de wants_enter cuando el coche va dirección sur, se verifican todas las condiciones de existencia (1 a 6), que inside_north = 0 e inside_south > 0, y que o bien el contador_south < MAX o bien waiting_north = 0 (porque se ha abierto el monitor).

Resulta análogo para la dirección norte, ya que es completamente simétrico.

LEAVES_TUNNEL()

```
def leaves_tunnel(self, direction):
```

{Como es la dirección opuesta a la que acabamos de comentar en wants_enter(), se verifican: todas las condiciones de existencia (1 a 6), que inside_north > 0 y que inside_south = 0, aunque como hay otros procesos en paralelo, no tiene por qué verificarse que contador_north < MAX o que waiting_south = 0 }

```
        self.mutex.acquire()
        if direction == NORTH:
                self.inside_north.value = self.inside_north.value - 1
```

Sabemos al entrar a leave_tunnel() que inside_north > 0, ya que siempre entramos después de un wants_enter(). Como cada vez que salimos de esta función sumamos 1 a inside_north y solo restamos 1 al salir de leave_tunnel, se tiene que inside_north > 0. Por tanto, si restamos se sigue verificando (1) inside_north >= 0.

```
                if (self.inside_north.value==0):
                        self.contador_south.value = 0
```

(6) contador_north >= 0 se verifica.

```
                        self.sem_north.notify_all()
                        self.sem_south.notify_all()
```

Si inside_north = 0, entonces pueden seguir pasando coches dirección norte si waiting_south = 0 o si counter_north < MAX. Si waiting_south > 0 y counter_north ≥ MAX, entonces podrán pasar del sur. De eso se ocupan los monitores, por eso notificamos a ambos.

Resulta análogo para la dirección contraria.