Túnel de Kiyotaki

Entrega de Programación Paralela

Adrián Lattes Grassi (alattes@ucm.es)

26 de Marzo de 2021

Organización del código

He organizado el código en clases y subclases y he separado las definiciones en ficheros distintos. De este modo he evitado repetir código (y de paso he repasado la herencia de clases en python).

Estos son los ficheros:

- `Car.py`: Clase para los coches.
- `TunnelCommon.py`: Clase base que implementa la parte común de los monitores.
- `Tunnel*.py`: Subclases de `TunnelCommon` donde se implementan las distintas versiones de los monitores. Estos archivos además son ejecutables.

Además he incluido docstrings en el código en las que intento explicar que hace cada cosa…

Solución naive: TunnelNaive.py

Esta es la solución más simple que se me ha ocurrido.

El monitor tiene un valor `_is_empty` que indica si hay un coche en el túnel y una condición para determinar si se puede entrar en el mismo. Para entrar en el túnel, la condición espera a que se pueda entrar, pasando a `wait_for` el método `can_enter`, que comprueba si `_is_empty` vale cero o no.

En esta solución no se tiene en cuenta la dirección de los coches. Por tanto, para evitar choques, solo se permite que haya un coche a la vez en el túnel.

Pros y contras

Lo bueno de esta solución es que es muy fácil de implementar y usa poca memoria, pero tiene muchos problemas asociados.

Por un lado, es evidente que no es nada eficiente que solo se permita que transite un coche por el túnel en simultáneo, puesto que los coches que vienen desde la misma dirección que el que está en el túnel deberían de poder pasar detrás de él.

Además, si por ejemplo los coches viajan en una dirección más rápido que en otra, es bastante probable que estos bloqueen a los que quieren entrar desde el otro lado.

Teniendo en cuenta la dirección de los coches: TunnelImproved.py

Ahora sí que se tiene en cuenta la dirección de los coches, por lo que pueden entrar varios coches a la vez en el túnel.

Para poder hacer esto he tenido que añadir dos atributos y modificar las implementaciones de los métodos.

- `_current_dir`: para saber en que dirección están transitando los coches.
- `_current_ncars`: para saber cuantos coches están transitando.

Pros y contras

Esta solución mejora mucho la anterior, puesto que ya pueden pasar varios coches a la vez. Pero todavía no se evita el bloqueo, puesto que puede suceder que una vez que están pasando coches en una dirección sigan entrando indefinidamente desde el mismo lado.

Dejando pasar a los coches por grupos: TunnelGroups.py

La idea de esta solución es dejar pasar juntos a todos los coches que estan esperando en un momento, pero no a los que llegan después de que estos hayan entrado al túnel. Una vez estos hayan salido, puede pasar el siguiente grupo.

Para implementar esto he introducido dos atributos y he modificado las implementaciones de los métodos.

- `_waiting_cars`: para saber cuantos coches están esperando de cada lado.
- `_current_max_ncars`: almacena el tamaño del grupo que tiene permiso para circular.

Un caso problematico es cuando no se forman colas y los grupos que van entrando son de tamaño 1, por lo que se bloquea el paso a los coches que vienen por detrás y se genera una situación similar a la de la *solución naive*. He evitado este problema introduciendo un tamaño máximo de coches solo si hay coches esperando en el otro lado (ver línea 82).

Pros y contras

Esta es la mejor de las soluciones que he propuesto, ya que evita que los coches de un lado se queden esperando indefinidamente debido a que siguen entrando coches continuamente desde el otro lado.

De todos modos se siguen teniendo bastantes limitaciones. Por ejemplo

Más ideas (no implementadas)

- Añadir casos de prueba más complejos y completos, que demuestren mejor los problemas de las distintas soluciones.
- Generar turnos, para garantizar que los grupos entran alternadamente.
- Introducir un reloj global, para garantizar que los turnos no sean demasiado largos. Si no me equivoco, para esto sería necesario usar técnicas de programación distribuida.
- Mejorar la simulación de los coches, para evitar alcances

Implementación real

Las simulaciones aquí presentadas son muy elementales, abstractas y lejanas de una implementación real. Para ejecutar sobre el terreno estas ideas habría que tener en cuenta muchos más detalles, e introducir *sensores* que midan los valores de las variables que estamos usando.

Por ejemplo habría que introducir sensores (cámaras, sensores de movimiento, etc…) para detectar cuantos coches están esperando de cada lado y cuando salen del túnel.

En los dos extremos del túnel habría que introducir semáforos o barreras para evitar que los coches entren cuando no deben.

Seguridad

Cuando se quieren llevar a la práctica programas que gestionen situaciones de riesgo es muy importante garantizar la máxima seguridad posible.

Respecto al software, se pueden utilizar técnicas formales para garantizar que el programa funcione correctamente.

Respecto al hardware se pueden tomar distintas medidas de precaución para evitar problemas, como tener sistemas de emergencias, como sensores secundarios, por si falla el sistema principal.