Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Sistemas Operativos

Grupo 06

2025 - 2

Implementación del problema: Intersección de caminos

Alumnos

Eric Ramírez Valdovinos (423095203)

Erick Nava Santiago (320298608)

Profesor

Ing. Gunnar Wolf

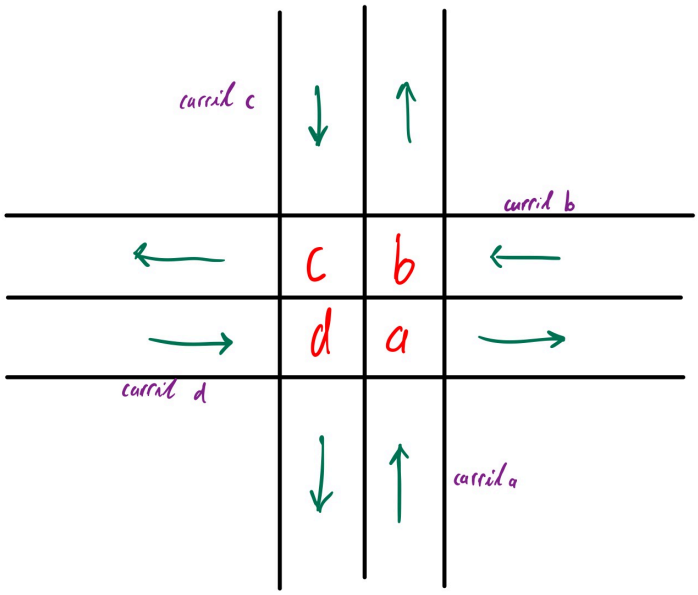
Ciudad Universitaria, Coyoacán, CDMX, 27 de marzo de 2025

Ejercicio de sincronización: Intersección de caminos

1. Descripción del Problema

Existe una intersección entre 4 carriles de doble circulación en la cual no hay semáforos, ni un tránsito para evitar accidentes. El problema se trata de simular el flujo de autos por la intersección implementando un mecanismo de sincronización entre los carros para que no existan colisiones entre ellos.
Teniendo en cuenta que el tráfico puede venir de cualquier lugar, en cualquier momento. ¿Cómo podemos evitar choques?

Los cuadrantes y carriles que se modelarán están representados a continuación:



1.1 Reglas

- No puede haber dos autos en la misma sección de la intersección a la vez (llamemos a esa situación accidente o choque), es decir que si la intersección se divide en cuatro cuadrantes, cada uno puede ser ocupado exclusivamente por un y solo un auto.
- No existe el rebase, los autos no invaden el carril izquierdo.
- No debes permitir que se lleve a la inanición: Aunque haya tráfico constante en un sentido, un auto que llegue desde otro debe poder cruzar. Los tiempos de espera serán admitidos, pero no indeterminados, un auto que esté esperando, eventualmente debe de poder pasar sin chocar.

2. Lenguaje y Entorno de Desarrollo

Lenguaje: Python

Entorno: Se desarrolló y testeó a través de VsCode utilizando el lenguaje Python y las bibliotecas threading, time y random.

No se requiere instalación de librerías externas.

El programa se dividió en tres archivos para facilitar la legibilidad.

2.1 Ejecución

Ubicar el programa en el directorio correspondiente, manteniendo la estructura de archivos y ejecutar el archivo main.py.

Ejecutar el programa mediante la línea de comandos:

python main.py

3. Estrategia de Sincronización

Para resolver el problema utilizamos semáforos para la intersección general y considerando cada uno de los 4 cuadrantes como un mutex de acceso exclusivo para un solo auto.

3.1 Mecanismo Utilizado:

Se emplearon locks para bloquear el acceso al recurso compartido, en este caso los cuadrantes de la intersección, de esta forma, cada auto representado por un hilo esperará su turno para acceder a las diferentes secciones de la intersección. Para ello se utilizaron las funciones acquire() y release().

Para representar a la intersección general usamos un semaphore(3) para permitir el acceso máximo a 3 autos a la vez en el cruce, esto para evitar la condición en la cual hasta 4 autos que quieran entran al mismo tiempo y se provoque una inanición en la cual ningún auto pueda avanzar en línea recta, siendo el caso en el que los 4 necesitan hacer movimientos distintos a doblar a la derecha.

3.2 Refinamientos Implementados

3.2.1 Evitar inanición

Como mencionamos anteriormente, se utilizó una condición de semáforo para considerar una entrada máxima de 3 autos a la intersección, esto sumado al uso de mutex para cada uno de los cuatro cuadrantes, nos permitió modelar un sistema de espera para los autos que quisieran entrar a uno de ellos, de tal manera que cuando un hilo de auto entrará a cierto cuadrante se adquiere el recurso con mutex.acquire() y en el momento que quisiera pasar o otro o salir de la intersección, usamos mutex.release() para liberarlo y adquirir otro o cerrar completamente el hilo con el auto.join(), respectivamente.

3.2.2 Permitir giros dentro de la intersección principal

En este caso, lo trabajamos de la manera más sencilla con las funciones de cambioCuadrante() y auto(). En esta última se genera un número aleatorio que modela si el auto lleva dirección para doblar a la derecha, seguir recto o doblar a la izquierda, lo cual implica un uso de 1 , 2 o 3 mutex totales para su

recorrido. En resumen, usamos una serie de condicionales para ir pidiendo acceso a las diferentes rutas de mutex de la cuadrícula en función de la ruta que se siga.
Ej: El auto a4 necesita doblar a la izquierda, para ello se adquiere el mutex_a y uno de los recursos del semáforo, utilizando acquire() para entrar a la intersección, seguido de esto, usamos la función cambioCuadrante() para adquirir el mutex_b (moverse hacia arriba), una vez este libre, lo adquirimos y liberamos el mutex_a (en ese orden para evitar colisiones), posteriormente hacemos el mismo proceso, pero para el mutex_C que se encuentra a la izquierda, adquiriendo el recurso y liberando el del mutex_b, finalmente, simulamos tiempos de espera para estos cruces y liberamos el mutex_C para expresar que el auto ha salido de la intersección y por lo tanto hacemos un release() de uno de los 3 espacios que permite adquirir el semáforo.

Se realizaron pruebas de sincronización para detectar posibles condiciones de carrera y se aplicaron correcciones en función de los resultados obtenidos.

4. Dudas y mejoras

La verdad es que nos quedamos cortos con respecto a la implementación de una interfaz gráfica que modele el fenómeno, porque el problema se prestaba bastante para mostrar cada estado de mutex y del semáforo en una ventana a la lado de una animación del cruce de autos, no obstante, no trabajamos lo suficientemente bien con respecto a los tiempo.

La función cambioCuadrante() pudo haber sido una implementación recursiva, lo cual haría del código más sencillo y elegante, pero una vez, decidimos quedarnos con la primera versión funcional.

Sería interesante ver la complejidad de la solución realizada al aplicar más carriles y una simulación de mayor peso en cuanto recursos utilizando más hilos. Me parece que la solución presentada no es la más eficiente, pero es funcional.

No se colocó un refinamiento que permitiría que los 4 autos se encontrarán en cada cuadrante, pues para que uno se moviera, otro debía de hacerlo. Esto al momento de sincronizar no era posible, puesto que si se movía uno, en automático se estarían chocando. Una propuesta es que se realizará un trabajo en paralelo, sin embargo, al trabajar con la línea de código no nos representó la posibilidad de representarlo, así mismo, el entendimiento de los refinamientos anteriores y la lógica usada en la parte de la programación no nos lo permitió hacer.