UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA

SISTEMAS OPERATIVOS

Profesora: Dr. Gunnar Eyal Wolf Iszaevich

Alumno: Echevarria Aguilar Luis Angel
No. Cuenta: 320236235

TAREA 1. Ejercicios de sincronización.

Semestre: 2026-1 **Fecha de entrega:** 16 / 10 / 2025

INFORME TÉCNICO DE ENTREGA

DESCRIPCIÓN GENERAL DEL PROBLEMA

El problema seleccionado para esta tarea es el de **"Intersección de Caminos"**. El desafío fundamental consiste en modelar un cruce de cuatro vías sin señalización, donde múltiples vehículos (representados como hilos de ejecución) intentan cruzar simultáneamente desde diferentes direcciones. El objetivo principal es diseñar un sistema de control que garantice la ausencia de choques (es decir, que dos vehículos no ocupen el mismo espacio al mismo tiempo) y, de manera crucial, que evite los interbloqueos (deadlocks), una situación en la que dos o más vehículos quedan permanentemente bloqueados, esperando el uno por el otro para poder avanzar. Cabe aclarar que no existe el rebase, por lo que los autos no pueden invadir el carril izquierdo y, sin importar la dirección de la que venga el auto, siempre se circula por el carril derecho.

Objetivos de sincronización:

- Evitar condiciones de carrera (race conditions)
- Prevenir bloqueos mutuos (deadlocks)

LENGUAJE, ENTORNO Y EJECUCIÓN

Lenguaje y Entorno: La solución fue desarrollada en Python 3.8 o superior. Se utilizó exclusivamente la biblioteca estándar threading, que viene incluida con cualquier instalación de Python, por lo que no se requieren dependencias externas. El uso de Python se eligió por su sintaxis clara y su modelo de hilos accesible, lo que permite centrarse en la complejidad de la lógica de sincronización en lugar de en la gestión de memoria o la configuración del compilador.

Instrucciones de Ejecución: Para ejecutar el programa, solo se necesita un intérprete de Python 3 instalado en el sistema. Abra una terminal o línea de comandos, navegue con el comando “cd” hasta el directorio donde se encuentra el archivo y ejecute el siguiente comando:

python inter_caminos.py

El programa iniciará la simulación, generando varios hilos que representarán a los autos. La salida en la terminal mostrará en tiempo real el estado de la intersección, indicando cuándo llegan los autos, cuándo adquieren los permisos para cruzar y cuándo liberan la intersección de forma segura.

ESTRATEGÍA DE SINCRONIZACIÓN

Para resolver este problema, se implementó una estrategia robusta basada en la división de recursos y un patrón anti-deadlock.

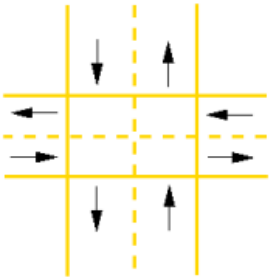
Mecanismo de Sincronización: Mutexes (Locks). Se modeló la intersección como un conjunto de cuatro recursos compartidos e independientes: los cuadrantes Noreste, Noroeste, Suroeste y Sureste. Para proteger cada uno de estos cuadrantes, se utilizó un Mutex (implementado como threading.Lock en Python). Un Mutex garantiza la exclusión mutua, lo que significa que solo un hilo (auto) puede "poseer" un cuadrante a la vez, previniendo eficazmente los choques.

Patrón Anti-Deadlock: Para evitar el deadlock, implemente una estrategia llamada Orden Jerárquico de Bloqueo (Hierarchical Lock Ordering). El simple hecho de usar un Mutex por cuadrante no es suficiente, ya que introduce un alto riesgo de deadlock. Para mitigar esto, se implementó el patrón de orden jerárquico de bloqueo. La estrategia consiste en:

1. **Asignar una jerarquía:** A cada Mutex de cuadrante se le asignó un identificador numérico único y fijo: 0 (NE - Noreste), 1 (NO - Noroeste), 2 (SO - Suroeste) y 3 (SE - Sureste).

2. **Imponer un Orden de Adquisición:** Se estableció una regla de oro para todos los hilos, donde independientemente de la trayectoria física del auto, los locks deben ser adquiridos siempre en orden ascendente según su identificador jerárquico.
   Siempre priorizará adquirir el lock en el siguiente orden NE -> NO -> SO -> SE.

3. **Cómo ocurre un deadlock:**



   - Auto 1 (Sur->Norte) bloquea el cuadrante SO y espera por el NO.
   - Auto 2 (Oeste->Este) bloquea el cuadrante NO y espera por el NE.
   - Auto 3 (Norte->Sur) bloquea el cuadrante NE y espera por el SE.
   - Auto 4 (Este->Oeste) bloquea el cuadrante SE y espera por el SO.

   ¡Todos esperan a otro en un ciclo mortal! Nadie avanza.
   La única forma de garantizar que no haya un ciclo de espera es que todos los hilos adquieran los mutexes en el mismo orden predefinido, sin importar su trayectoria.

4. **Ejemplo práctico:** Un auto que viaja del Sur al Este necesita los cuadrantes SO (ID 2), SE (ID 3) y NE (ID 0). Aunque su ruta física es 2 -> 3 -> 0, el programa primero determina la lista de locks necesarios [0, 2, 3], la ordena [0, 2, 3] y luego procede a adquirirlos en ese orden estricto. Al forzar a todos los hilos a solicitar los recursos en la misma secuencia global, se rompe la condición de "espera circular" que es necesaria para que ocurra un deadlock, garantizando que el sistema siempre pueda progresar.

IMPLEMENTACIÓN DE REFINAMIENTOS

Esta solución incorpora de manera integral los dos refinamientos propuestos en el planteamiento del problema.

1. **Mejora de Eficiencia (Refinamiento 1):** En lugar de bloquear toda la intersección con un único lock, la estrategia de dividirla en cuatro cuadrantes con Mutexes individuales permite un mayor grado de concurrencia. Por ejemplo, un auto girando a la derecha desde el Sur hacia el Oeste (usando solo el cuadrante SO) puede hacerlo al mismo tiempo que otro auto

gira a la derecha desde el Norte hacia el Este (usando solo el cuadrante NE), ya que no compiten por los mismos recursos.

2. **Modelado de Giros (Refinamiento 2):** La lógica central del programa está diseñada para manejar trayectorias complejas que requieren la adquisición de uno, dos o tres cuadrantes (giros a la derecha, seguir de frente y giros a la izquierda). Es precisamente esta complejidad la que hace necesaria y relevante la estrategia de orden jerárquico de bloqueo para asegurar un funcionamiento correcto y sin bloqueos.

   Giro a la derecha: Necesita 1 cuadrante. (Ej: Del Sur al Oeste, necesita el cuadrante SO).
   Seguir de frente: Necesita 2 cuadrantes. (Ej: Del Sur al Norte, necesita SO y NO).
   Giro a la izquierda: Necesita 3 cuadrantes. (Ej: Del Sur al Este, necesita SO, SE y NE).

OBSERVACIONES

Cuando un auto (hilo) intenta adquirir un lock que está ocupado, el sistema operativo lo pone en una cola de espera para ese lock específico. Sin embargo, cuando el lock se libera, no hay una garantía estricta de que el primer hilo que llegó a la cola sea el primero en ser despertado. El planificador de hilos del sistema operativo toma esa decisión para optimizar el uso general del CPU, por lo que el orden puede no ser estrictamente FIFO (First-In, First-Out).

Observe que el sistema no tiene una prioridad global por orden de llegada. La prioridad real es una combinación de disponibilidad de recursos y el orden jerárquico de bloqueo. Los autos con rutas más simples que requieren cuadrantes menos competidos pueden cruzar mucho más rápido, sin importar cuándo llegaron.

CONCLUSIONES

La simulación desarrollada para el problema de la intersección de caminos demuestra de manera efectiva la aplicación de mecanismos de sincronización para resolver un problema de concurrencia clásico. La implementación cumple exitosamente con los dos objetivos de seguridad fundamentales: garantiza la exclusión mutua en cada cuadrante, previniendo colisiones, y elimina la posibilidad de interbloqueo (deadlock) mediante una estrategia de orden jerárquico de bloqueo. Se observa que el orden en que los vehículos cruzan no sigue necesariamente su orden de llegada, sino que es un comportamiento emergente dictado por la competencia de recursos y la lógica del planificador de hilos del sistema operativo. Los vehículos con rutas más simples o que requieren cuadrantes menos disputados a menudo pueden proceder más rápidamente, lo que refleja una característica realista de los sistemas con recursos limitados.

A pesar de su robustez en cuanto a seguridad, el análisis de la implementación revela áreas donde su comportamiento podría refinarse para mejorar aspectos como la equidad y la eficiencia general. Una de las principales observaciones es la posibilidad teórica de inanición (starvation). En escenarios de tráfico denso y desbalanceado, un vehículo con una ruta compleja que requiere múltiples cuadrantes muy solicitados podría ser pospuesto indefinidamente, mientras otros vehículos con necesidades más simples lo adelantan repetidamente. El modelo actual no implementa un mecanismo de equidad explícito para mitigar esta situación.

Otra área de mejora se relaciona con el rendimiento (throughput) del sistema. La estrategia de orden jerárquico, si bien es infalible para prevenir deadlocks, puede ser restrictiva. Obliga a un vehículo a adquirir locks en un orden predefinido, lo que podría significar bloquear un cuadrante mucho antes de que sea físicamente necesario, impidiendo que otros vehículos lo utilicen. Para maximizar el flujo de vehículos, se podrían explorar algoritmos más avanzados que relajen estas restricciones, como sistemas de detección y recuperación de deadlocks o un "controlador de tráfico" centralizado que asigne permisos de paso de una manera más holística y optimizada.

Finalmente, el realismo de la simulación podría aumentarse introduciendo variables como diferentes velocidades entre los vehículos, tiempos de aceleración y deceleración, y la posibilidad de que no todos los cuadrantes estén ocupados por el mismo intervalo de tiempo. Estas mejoras, aunque van más allá del problema de sincronización principal, permitirían modelar un escenario de tráfico de manera más fiel a la realidad.