UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ingeniería

Sistemas Operativos

Tarea 1. Ejercicios de Sincronización

Profesor: Dr. Gunnar Eyal Wolf Iszaevich

Grupo: 8

Integrantes:

- Alvarez Salgado Eduardo Antonio (321335630)
- Morales Castillo Arumy Lizeth (321082626)

Fecha de entrega: 16 de octubre de 2025

Semestre: 2026-1

Problema a resolver: El cruce del río

Para llegar a un encuentro de desarrolladores de sistemas operativos, los asistentes deben cruzar un río en una sola balsa. En el evento participan dos tipos de desarrolladores:

- Hackers: desarrolladores de Linux
- Serfs: desarrolladores de Microsoft

La balsa tiene una capacidad exacta de cuatro personas. Por seguridad y convivencia, se deben cumplir las siguientes condiciones: La balsa no puede zarpar con menos de cuatro personas (si lo hace, se volcaría). La balsa solo puede transportar combinaciones seguras:

- Cuatro hackers (4H)
- Cuatro serfs (4S)
- Dos hackers y dos serfs (2H + 2S)

No se permite subir tres de un tipo y uno del otro (3 Hackers + 1 Serf o 3 Serf + 1 Hacker). Solo hay una balsa, y regresa automáticamente después de cada viaje. El objetivo es sincronizar correctamente la subida y zarpado de la balsa, evitando condiciones de carrera y cumpliendo las reglas de balance.

Solución

La solución propuesta está implementada en un programa escrito en lenguaje de programación Python, utilizando un sistema de control concurrente basado en semáforos para garantizar que solo combinaciones válidas de pasajeros crucen el río en la balsa.

1. Llegada de un desarrollador: Cuando llega un desarrollador (sea hacker o serf), se incrementa el contador correspondiente (num_hackers o num_serfs). Esta operación se realiza dentro de un mutex (mutex.acquire()), lo que asegura que los contadores no se modifiquen de forma concurrente por varios hilos al mismo tiempo. De esta manera se evitan condiciones de carrera.

2. Formación de la balsa: Dentro de la sección crítica, el hilo que detecta que se cumplió una combinación válida de 4 pasajeros decide la conformación del viaje. Las combinaciones permitidas son:

- 4 hackers
- 4 serfs
- 2 hackers + 2 serfs

En cualquiera de estos casos, se liberan exactamente 4 permisos en los semáforos correspondientes (sem_hackers y/o sem_serfs). Esto permite que los hilos seleccionados puedan continuar y abordar la balsa.

A diferencia de un sistema aleatorio, aquí la formación del grupo depende de la llegada y acumulación ordenada de los desarrolladores, no del azar.

3. Uso de semáforos diferenciados: Los semáforos sem_hackers y sem_serfs actúan como colas de espera separadas. Cada desarrollador que llegó se bloquea en su semáforo hasta que haya sido elegido como parte de una combinación válida. De este modo se asegura que:

- Solo los desarrolladores necesarios suban.
- Se respeten las proporciones de cada grupo.

4. Abordaje de la balsa: Una vez liberado su semáforo, cada desarrollador imprime el mensaje de abordaje y aumenta el contador num_abordando. Este contador permite verificar que exactamente 4 hilos hayan subido antes de que la balsa pueda zarpar.

- El último en abordar (cuando num_abordando == 4) es el responsable de:
- Imprimir que la balsa parte.
- Simular el viaje (time.sleep(1)).
- Mostrar la composición del grupo que cruzó el río (ejemplo: “2 Hackers y 2 Serfs”).

5. Reinicio de contadores y mutex: Tras el cruce, se reinician los contadores de la balsa (num_abordando, hackers_abordando, serfs_abordando). En este momento también se libera el mutex, el cual había permanecido bloqueado desde la formación de la balsa.

Este detalle es clave ya que garantiza que no se inicien dos balsas al mismo tiempo ni se mezclen desarrolladores de diferentes viajes.

6. Repetición infinita: El programa genera desarrolladores de manera aleatoria en un ciclo infinito, por lo que la formación de balsas válidas se repite continuamente. Siempre que haya hilos acumulados, se formará una nueva balsa y se repetirá el proceso.

Consideraciones

No se solicitó ningún refinamiento adicional, por lo que el código se limita a resolver la problemática inicial utilizando mecanismos de sincronización.

Algo que se podría mejorar es implementar alguna condición o mecanismo para que finalice la ejecución, ya que se tiene un bucle infinito que genera de manera aleatoria hilos, los cuales simulan la llegada de un desarrollador al río y si no se tiene mucho cuidado, la simulación puede consumir muchos recursos.