Universidad Nacional Autónoma de México

Facultad de Ingeniería

Sistemas Operativos

Grupo #06

Tarea #01: Ejercicios de sincronización

Miyasaki Sato Yuichi Vicente

No. Cuenta 318586465

Fecha de entrega: 22/10/24

Profesor: Ing. Gunnar Eyal Wolf Iszaevich

Contenido

1.- Problema propuesto: El cruce del rio.

1.1.- Descripción del problema.

El desafío de problema consiste en ayudar a un grupo diverso de personas, que incluye hackers y serfs, a cruzar un río utilizando una balsa con capacidad limitada. Para asistir a un encuentro de desarrolladores de sistemas operativos, es fundamental que logren cruzar sin complicaciones ni conflictos.

El objetivo es diseñar una solución que garantice el cruce seguro de todos los participantes, evitando cualquier tipo de confrontación. Para lograrlo, se implementarán estrategias que permitan abordar la balsa en diversas configuraciones. Además, se simularán acuerdos en caso de que surjan conflictos, asegurando así que todos crucen el río de manera ordenada y pacífica.

1.2.- Reglas.

- En la balsa caben cuatro (y sólo cuatro) personas.
- La balsa es demasiado ligera, y con menos de cuatro puede volcar.
- Al encuentro están invitados hackers (desarrolladores de Linux) y serfs (desarrolladores de Microsoft).
- Para evitar peleas, debe mantenerse un buen balance: No debes permitir que aborden tres hackers y un serf, o tres serfs y un hacker. Pueden subir cuatro del mismo bando, o dos y dos.
- Hay sólo una balsa.
- No se preocupen por devolver la balsa (está programada para volver sola).

2.- Lenguaje y entorno de desarrollo.

2.1.- Entorno de ejecución.

El programa fue desarrollado en Python, y se puede ejecutar directamente en Visual Studio Code o en la terminal. Además, se entregará un archivo .exe para que los usuarios puedan ejecutarlo directamente.

2.2.- Librerías utilizadas.

Se emplearon varias librerías de Python que facilitan la implementación de funciones clave, que son:

- Threading: Para crear y manejar múltiples hilos.
- Random: Para generar números y eventos aleatorios (o pseudoaleatorios).
- Time: Para funciones que manipulen el tiempo, ya sea crear pausas o simular acciones.
- Colorama: Para imprimir texto en diferentes colores en la consola.

Para hacer uso de la librería Colorama ejecutaremos los siguientes comandos en la consola:

- python -m pip install colorama

Es necesario inicializar en código la siguiente sentencia:

- from colorama import init, Fore, Style
- init()

3.- Estrategia de sincronización.

3.1.- Semáforos.

Se utiliza un semáforo con un contador inicial de 4 para limitar el número de personas que pueden acceder a la balsa simultáneamente. Este semáforo garantiza que no más de 4 personas puedan estar en la balsa al mismo tiempo.

Con acquire se llama para que un hilo intente entrar en la balsa, y con release se llama para permitir que otros hilos accedan a la balsa después de que un hilo ha terminado su acción.

3.2.- Mutex.

Se utiliza un mutex para garantizar que solo un hilo pueda acceder a ciertos recursos compartidos a la vez, en el caso específico para proteger el acceso a la lista balsa_list y los contadores de hackers y serfs.

También asegura que el bloque de código que modifica el estado de la balsa solo puede ser ejecutado por un hilo a la vez.

El uso de los mutex evita condiciones de carrera donde varios hilos intentan modificar la misma lista simultáneamente, en este caso la balsa.

3.3.- Clases principales del programa.

Persona: Clase base que representa a una persona que intentará subir a la balsa. Tiene un método run() que hace que la persona intente subir a la balsa indefinidamente.

Hacker: Hereda de Persona y representa a un hacker. Su método representar() devuelve un emoji visual.

Serf: También hereda de Persona y representa a un serf, con un método representar() similar.

Balsa: Maneja la lógica del cruce del río usando semáforos, mutex y contadores.

3.4.- Funciones principales del programa.

Subir: Intenta que una persona suba a la balsa, esto mediante un semáforo para limitar el acceso. Con el mutex, agrega a la persona a la lista y actualiza los contadores; cuando hay 4 personas en la balsa, llama al método zarpar().

Zarpar: Verifica si puede zarpar, si hay un conflicto realiza un acuerdo entre bandos con la función simular_acuerdo(), y si no hay conflictos sigue el programa y reinicia los contadores.

Crear personas: Genera personas de manera aleatoria y las hace subir a la balsa, simulando variaciones en sus tiempos de llegada, ya sean hackers o serfs.

4.- Refinamientos.

Para mi problema no existía refinamiento alguno, pero implemente una condición cuando surgen conflictos entre los bandos.

Esta condición establece que, si hay 3 hackers con 1 serf o 3 serfs con 1 hacker, simula un proceso de acuerdo entre bandos. Esto permite que los grupos lleguen a un consenso y crucen el río en diferentes configuraciones, cabe aclarar que las decisiones que tomes los bandos es aleatoria.

5.- Dudas y mejoras.

Una de las mejoras que me gustaría implementar es la gestión de conflictos entre los bandos a través de un sistema de votación en lugar de seleccionar aleatoriamente a los participantes. Esta función permitiría que los hilos (personas) voten por los hackers y serfs que deben abordar la balsa, haciendo el proceso más democrático y transparente. De este modo, se podrán mostrar las elecciones realizadas por los participantes, reflejando quiénes fueron los seleccionados para cruzar el río, lo que añadiría una capa adicional de interacción y colaboración al programa.