UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Faculta de ingeniería

Sistemas Operativos

Grupo: 06

Semestre 2026-1

-Tejeda Vaca Abraham: 321328456

Tarea 01. Ejercicios de sincronización

Fecha de entrega: 16/Octubre/2025

1. **El problema que decidí resolver** 

Resolveré el problema clásico de concurrencia “El cruce del río”. Hay dos tipos de personas, hackers (H) y serfs (S), que deben cruzar un río usando una balsa con capacidad para 4 pasajeros. Para evitar conflictos, la balsa solo puede zarpar con alguna de estas combinaciones válidas:

- 2 hackers y 2 serfs (2H + 2S)
- 4 hackers (4H)
- 4 serfs (4S)

Las llegadas de personas son concurrentes (independientes y potencialmente simultáneas). El objetivo es sincronizar el abordaje para que la balsa salga exactamente con 4 pasajeros, respetando las combinaciones permitidas y evitando condiciones de carrera o esperas activas (*busy waiting*).

Requisitos y supuestos principales:

- La decisión de qué grupo embarca se toma de forma atómica bajo exclusión mutua (para proteger los contadores de H y S).
- La balsa solo zarpa cuando hay 4 listas (barrera de embarque).
- Si existe mezcla posible (2H + 2S), se prioriza esa opción para promover equidad y evitar que un solo tipo “acapare” los viajes.
- La sincronización se realizará con semáforos, locks y barrera, sin sondeo activo.

2. **Lenguaje y entorno** 

- Lenguaje: Python 3 (probado con Python 3.10+).
- Sistema: funciona en Linux, macOS y Windows.
- Dependencias externas: ninguna (solo librerías estándar de Python).

Miniguía para ejecutarlo. 

1) Requisitos

- Python 3.10+ (probado con 3.11).
- No hay dependencias externas (solo threading, time, random, argparse del estándar).

2) Obtener el código

Guarda el archivo como, por ejemplo, cruce_del_rio.py en una carpeta de tu preferencia.

3) Crear un entorno virtual (Opcional)

Windows (PowerShell):

```
PS C:\Users\abrah> python -m venv .venv
>> .venv\Scripts\Activate.ps1
```

Linux/macOS (bash/zsh):

```
abrah@DESKTOP-4NAE9D2 MINGW64 ~
$ python3 -m venv .venv
source .venv/bin/activate
```

(No hay nada que instalar con pip. El entorno virtual es opcional.)

5) Parámetros útiles (CLI)

- --personas N → total de personas a simular (por defecto 20).
- --p_hacker P → probabilidad de que una persona sea H (0–1, por defecto 0.5).
- --seed S → semilla para reproducibilidad (entero).

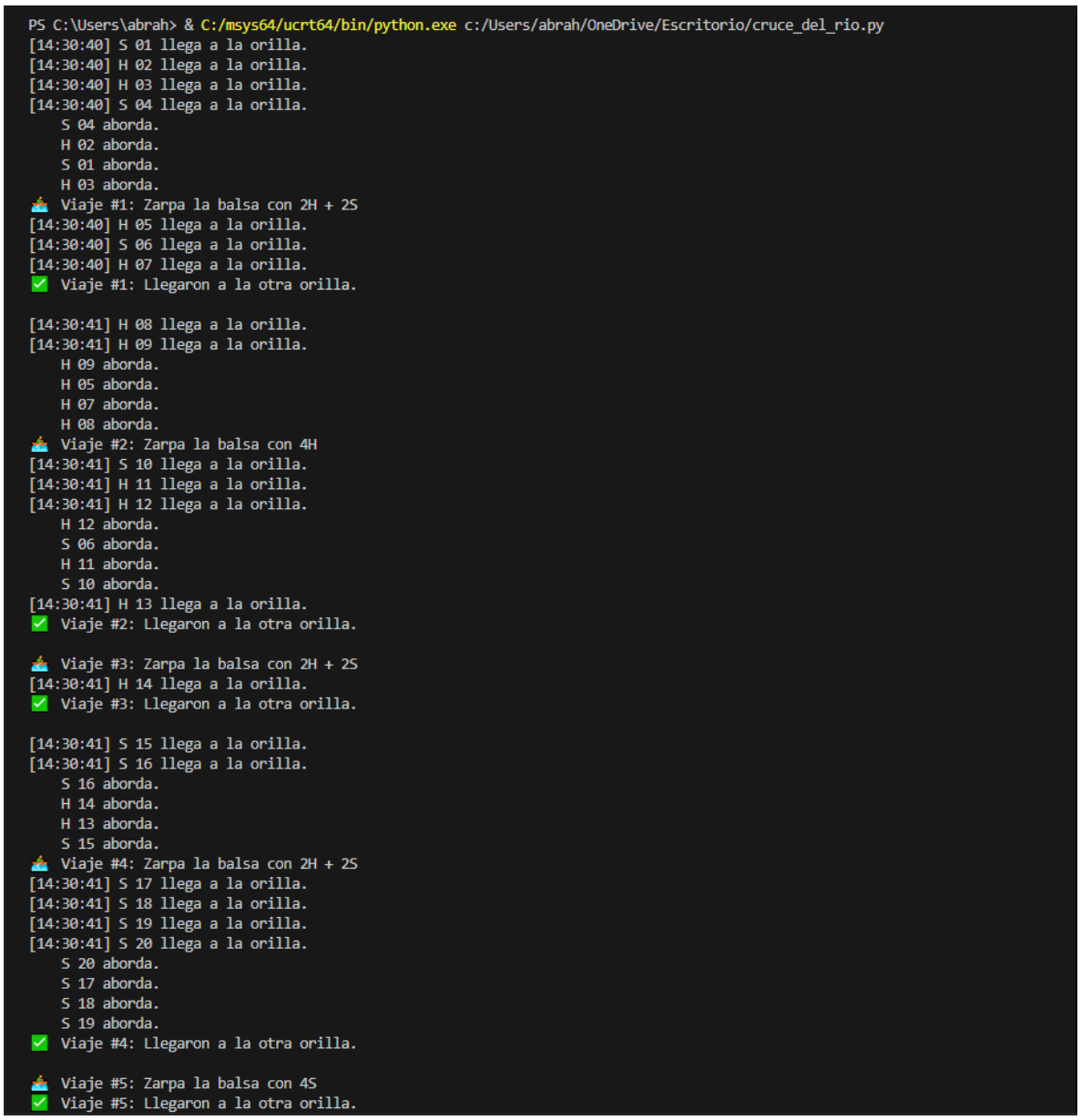
Ejemplos:

```
# 40 personas, proporción balanceada, corrida reproducible
>> python cruce_del_rio.py --personas 40 --p_hacker 0.5 --seed 123
>>
>> # Escenario "dominan hackers" (verás viajes 4H más seguido)
>> python cruce_del_rio.py --personas 40 --p_hacker 0.9
>>
>> # Escenario "dominan serfs"
>> python cruce_del_rio.py --personas 40 --p_hacker 0.1
>>
```

6) ¿Qué debo ver en la salida?

- Mensajes de llegada y abordaje intercalados (propio de la concurrencia).
- Por cada 4 pasajeros:
  - 🚣 Viaje #k: Zarpa la balsa con …
  - ✅ Viaje #k: Llegaron a la otra orilla.
- Nunca aparecerán viajes 3+1; sólo 2H+2S, 4H o 4S.
- Para N personas, deberías ver exactamente N/4 viajes.

Ejemplo de ejecución normal (por defecto):



Deberías observar únicamente viajes 4H/4S/2H+2S (sin 3+1).

3. Funcionamiento del código. 

Objetivo: garantizar que la balsa zarpe sólo con 4 personas en combinaciones válidas (2H+2S, 4H o 4S), evitando 3+1 y con una sola balsa.

- Recurso: Lock (self.mutex).
- Región crítica protegida:
    1. Actualización de contadores globales hackers_esperando y serfs_esperando.
    2. Decisión atómica del siguiente grupo a embarcar (política: preferir 2H+2S; si no, 4 del mismo bando).
    3. Publicación del grupo elegido mediante semáforos (ver punto 2).
- Qué garantiza:
    - No hay condiciones de carrera en contadores ni en la elección del grupo.
    - La decisión de grupo es indivisible (nadie interfiere entre “calcular” y “publicar” cupos).

Semáforos (control de acceso por tipo)

- Recursos: Semaphore por tipo (fila_hackers, fila_serfs), inicializados en 0.
- Uso:
    - La decisión hecha bajo mutex “abre lugares” con release() (p.ej., 2 a H y 2 a S).
    - Cada persona bloquea con acquire() en la fila de su tipo hasta que haya cupo.
- Qué garantiza:
    - Suben exactamente los hilos autorizados por la decisión atómica.
    - No hay sondeo (*no* while cond: pass): el hilo duerme hasta tener permiso.

    - Se evita 3+1 porque sólo se liberan 2/2 o 4/0 o 0/4 plazas.

Barrera (punto de encuentro de 4)

- Recurso: Barrier(4, action=_accion_de_zarpe).
- Uso:
    - Tras “abordar”, cada hilo llama a wait().
    - Cuando llegan 4, la barrera ejecuta una única action: imprime el manifiesto y simula el cruce.
- Qué garantiza:
    - Zarpe atómico de 4 personas (ni más ni menos).
    - Un solo “capitán” lógico por viaje (la action corre una sola vez).
    - Una sola balsa en curso: los siguientes grupos se forman y esperan su propia barrera.

Invariantes y propiedades

- Seguridad (safety):
    - Nunca hay viajes 3+1 → los semáforos sólo liberan 2H+2S o 4 iguales.
    - No hay solapamiento de grupos → la decisión se hace bajo mutex y la salida se sincroniza con barrera.
- Viveza (liveness):
    - Si hay suficientes personas para alguna combinación válida, eventualmente se publica y el grupo progresa (los hilos dejan de esperar al recibir release()).
    - No hay deadlock: los únicos bloqueos son en acquire() (espera de cupo real) y Barrier.wait() (punto de reunión de 4), ambos liberados por eventos del propio sistema.
- Fairness (simple):
    - Política “mezcla primero” (2H+2S) para evitar acaparamiento cuando hay de ambos tipos.
    - Si no hay mezcla posible, se permite 4 iguales para no frenar el sistema.
    - (Opcional) Puede añadirse alternancia estricta si el profesor pide anti-starvation fuerte.

Flujo resumido por hilo

1. Llega → entra a mutex → incrementa contador de su tipo.
2. Bajo mutex, si hay quórum, publica cupos con semáforos y registra el manifiesto del viaje.
3. Espera su semáforo (bloqueo pasivo, sin sondeo).
4. Aborda → Barrier.wait() → la acción de la barrera realiza el zarpe y la llegada.

4. Refinamientos implementados

- Preferencia por mezcla (fairness simple): si hay personas suficientes de ambos tipos, el sistema prioriza 2H+2S antes que 4 iguales. Esto reduce acaparamiento de un solo bando sin detener el sistema cuando sólo hay un tipo disponible.

- Zarpe atómico con Barrier(action=...): el “capitán” lógico del viaje es la acción de la barrera, que se ejecuta una sola vez cuando llegan los 4. Evita impresiones duplicadas y garantiza un único evento de zarpe/llegada por viaje.

- Manifiesto protegido por lock dedicado: el ID de viaje y su composición se guardan bajo viaje_mutex para que el reporte sea consistente incluso con intercalado de hilos.

- Ejecución finita y reproducible: parámetros CLI (--personas, --p_hacker, --seed) para controlar tamaño del experimento, sesgo de tipos y reproducibilidad.