Proyecto 1 - Concurrencia

Integrantes:

- **Torres Delgadillo Samuel Mixcoatl**
- **Montiel Juárez Oscar Iván**

Descripción:

En este proyecto nos centrarmos en el estacionamiento de la Facultad de Ingeniería, donde se han identificado diversos inconvenientes debido al mal manejo del funcionamiento y sistema. Entre los problemas se destacan:

- El estacionamiento se llena muy rápidamente (alrededor de las 6:30), lo que dificulta encontrar cupos durante el resto del día.
- Se permite un sobrecupo de boletos; por ejemplo, en un estacionamiento con capacidad para 500 carros, se ingresan 510, lo que genera confusión interna.
- La liberación de espacios provoca competencia entre vehículos: al liberarse un espacio, varios carros pueden intentar ocuparlo simultáneamente, lo que puede ocasionar choques entre coches y si un carro no es capaz de estacionarse en un largo tiempo puede llegar tarde o perder la clase.
- Se presentan casos en los que algunos vehículos se estacionan de manera incorrecta, ocupando más de un lugar, por lo que la grúa tiene que intervenir para llevarse el carro

Consecuencias nocivas de la concurrencia

Si no se controlan adecuadamente los accesos y liberaciones, se pueden derivar de las siguientes situaciones:

- Condición de carrera: Dos o más vehículos detectan un espacio libre al mismo tiempo y compiten por él, generando conflictos lógicos en la simulación.
- Inanición: Algunos vehículos pueden quedar esperando excesivamente, lo que puede provocar que pierdan clases o se generen largas esperas dentro del estacionamiento.

- Uso incorrecto de los recursos: Vehículos que se estacionan mal y ocupan más de un espacio, afectando la disponibilidad real de cupos.

Eventos a controlar y ordenamiento relativo Eventos críticos a controlar:

- Coordinación en la toma de espacios: Se debe regular la asignación de un espacio cuando se libera para evitar condiciones de carrera.
- Notificación de acceso denegado: Informar a los usuarios cuando no se pueda ingresar al estacionamiento.
- Manejo de vehículos mal estacionados: Detectar y gestionar situaciones en las que un vehículo ocupa más de un espacio.

Eventos en los cuales el orden relativo no es importante:

- La secuencia en que se liberan los espacios no es crítica, ya que lo fundamental es coordinar adecuadamente su ocupación una vez disponibles.

Propuesta de implementación

El proyecto se implementará en Python utilizando el módulo threading para el manejo de hilos. Se hará uso de la librería rich para generar una interfaz de usuario en la consola que muestre en tiempo real:

- Una tabla que diferencie y muestre los espacios libres y ocupados para motos y carros.
- Un listado actualizado de los vehículos en espera.
- Un registro de eventos (por ejemplo: "Carro 3 ocupó espacio A1" o "Carro 7 liberó espacio C7").

Mecanismos de sincronización

Para garantizar la correcta concurrencia y evitar problemas como condiciones de carrera o inanición se implementarán los siguientes mecanismos:

a) Semáforos

- Se utilizarán para controlar el número máximo de vehículos que pueden ingresar al estacionamiento, limitando efectivamente el acceso conforme a la capacidad definida.

```
semaforo = threading.Semaphore(sobreCupo)
```

b) Mutex (Exclusión Mutua)

- Se implementarán mutex para asegurar que las secciones críticas (como la actualización de la tabla de espacios o el registro de eventos) sean accedidas por un solo hilo a la vez, evitando inconsistencias en el estado compartido.

```
mutexEspacios = threading.Lock()
```

c) Variables de condición (opcional, para mejorar la coordinación)

- Se pueden emplear variables de condición para que los hilos de vehículos esperen de forma eficiente la notificación de que un espacio se ha liberado. Esto permite que, en vez de estar en un bucle activo (busy waiting), los hilos se bloqueen y sean llamados únicamente cuando se cumpla la condición necesaria (la disponibilidad del espacio).

```
condicionEspacio = threading.Condition(mutexEspacios)
```

Lógica de Operación de cada Hilo

- **Vehículos**:

1. Inicio:
   - Genera un ID único.
   - Intenta adquirir permiso del semáforo (1 o 2 permisos si se estaciona mal) (`semaforo.acquire()`).
2. Asignación de espacios:
   a) Si es un estacionamiento bien realizado:
   - Adquiere 1 permiso del semáforo.
   - Bloquea mutexEspacios.
   - Busca un espacio libre en el diccionario espacios (usando `condicionEspacio.wait()` si no hay).

    - Si encuentra: lo ocupa y actualiza asignacionVehiculos.
    - Si no: espera en condicionEspacio.wait() hasta ser notificado.
  - Libera mutexEspacios.

  b) Si es un estacionamiento mal realizado:

  - Adquiere 2 permisos del semáforo.
  - Busca dos espacios consecutivos libres.
    - Si encuentra: los ocupa.
    - Si no: libera los permisos y lanza excepción.

3. Estancia:
   - Simula un tiempo aleatorio de estacionamiento (3-7 segundos).
4. Salida:
   - Libera el/los espacios ocupados (en bloque `finally`).
   - Devuelve los permisos al semáforo.
   - Notifica a todos los hilos en espera (condicionEspacio.notify_all()).

- **Grúa**:

1. Inicio:
   - Bucle infinito hasta que se active stop_event.
2. Operación:
   - Espera 7 segundos.
   - Bloquea mutexEspacios.
   - Busca vehículos mal estacionados en asignacionVehiculos.
     - Si encuentra: libera sus espacios y actualiza el estado.
   - Libera mutexEspacios.
   - Notifica a los hilos en espera (`condicionEspacio.notify_all()`).

- **Interfaz de usuario**

1. Inicio:

- Usa la biblioteca Rich para mostrar una tabla en tiempo real.

2. Actualización:
   - Cada 0.3 segundos:
     - Bloquea mutexEspacios.
     - Calcula espacios libres/ocupados.
     - Obtiene los últimos eventos de registroEventos.
     - Actualiza la tabla.
     - Libera mutexEspacios.

Interacción entre Hilos

a) Vehículo ↔ Grúa

- **Escenario**: La grúa retira un vehículo mal estacionado.
  - La grúa libera 2 permisos del semáforo y 2 espacios en el diccionario.
  - Notifica a todos los vehículos en espera (`condicionEspacio.notify_all()`), lo que les permite reintentar ocupar espacios.

b) Vehículo ↔ Vehículo

- **Escenario**: Dos vehículos compiten por un espacio liberado.
  - Cuando un vehículo libera un espacio, notifica a los demás (`condicionEspacio.notify_all()`).
  - Los vehículos en espera despiertan y compiten por el mutex (`mutexEspacios`) para acceder al recurso. Solo uno gana y ocupa el espacio.

c) Interfaz ↔ Vehículos/Grúa

- **Escenario**: La interfaz muestra el estado actual del estacionamiento.
  - La interfaz adquiere `mutexEspacios` para leer `espacios` y `registroEventos`, garantizando consistencia en la visualización.

Estado Compartido

Variable/Recurso	Descripción	Mecanismo de Sincronización
`espacios`	Diccionario con el estado (ocupado/libre) de cada plaza (Ejemplo: `"A1": True`).	Protegido por `mutexEspacios`.
`asignacionVehiculos`	Diccionario que mapea ID de vehículos a plazas ocupadas (Ejemplo: `5: ("A3", "A4")`).	Protegido por `mutexEspacios`.
`registroEventos`	Lista de strings con eventos recientes (Ejemplo: `"Carro 12 ocupó A7"`).	Protegido por `mutexEspacios`.
`semaforo`	Semáforo contador que limita el número total de vehículos (incluyendo sobrecupo).	Operaciones atómicas (`acquire/release`).

Entorno de Desarrollo

- **Python**: 3.10.6

Bibliotecas:

- **Rich**: versión 12.6.0 o superior para la interfaz gráfica en terminal
- **Sistema Operativo**: Probado en Windows 10.

Módulos Clave:

- **threading**: Para gestión de hilos (vehículos, grúa, interfaz).
- **random**: Simula comportamientos aleatorios (tiempos de estancia, elección de espacios).
- **time**: Control de retardos y actualizaciones periódicas.

Requisitos para ejecutar

1. Abrir la terminal del sistema y dirigirse a la carpeta en la que se encuentre guardado el archivo “MontielOscar-TorresSamuel.py” mediante:

```
cd "[Direccion del archivo]" //Ejemplo: cd "Users/Yo/Downloads"
```

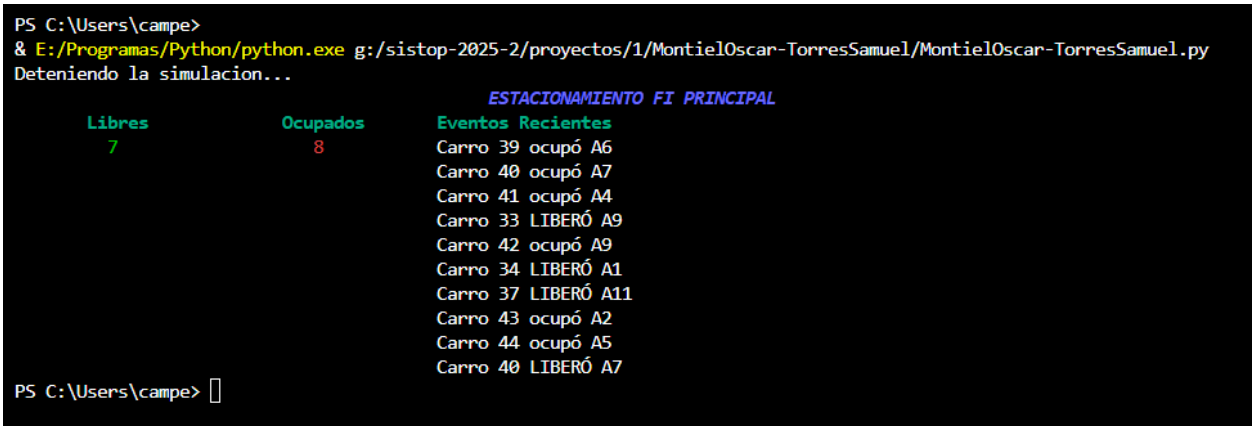
2. Instalar rich

```
pip install rich
```

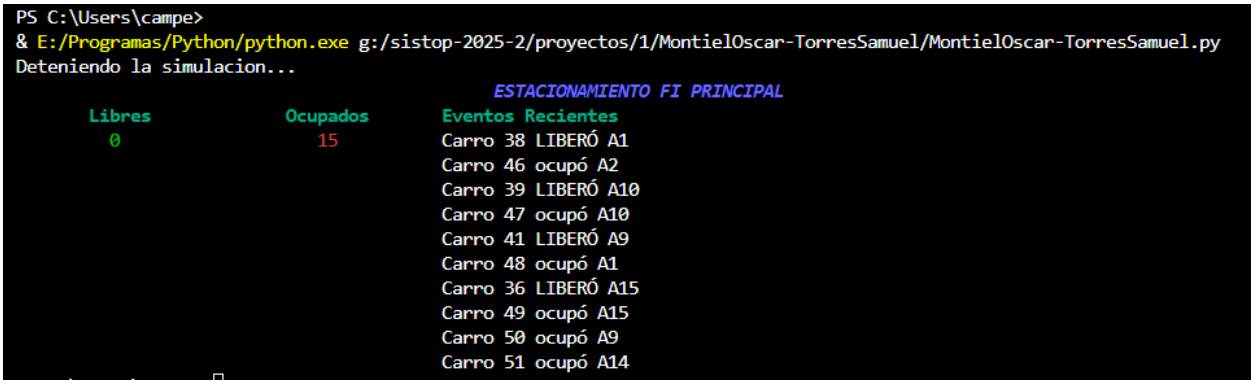
3. Ejecutar el programa

```
python MontielOscar-TorresSamuel.py
```

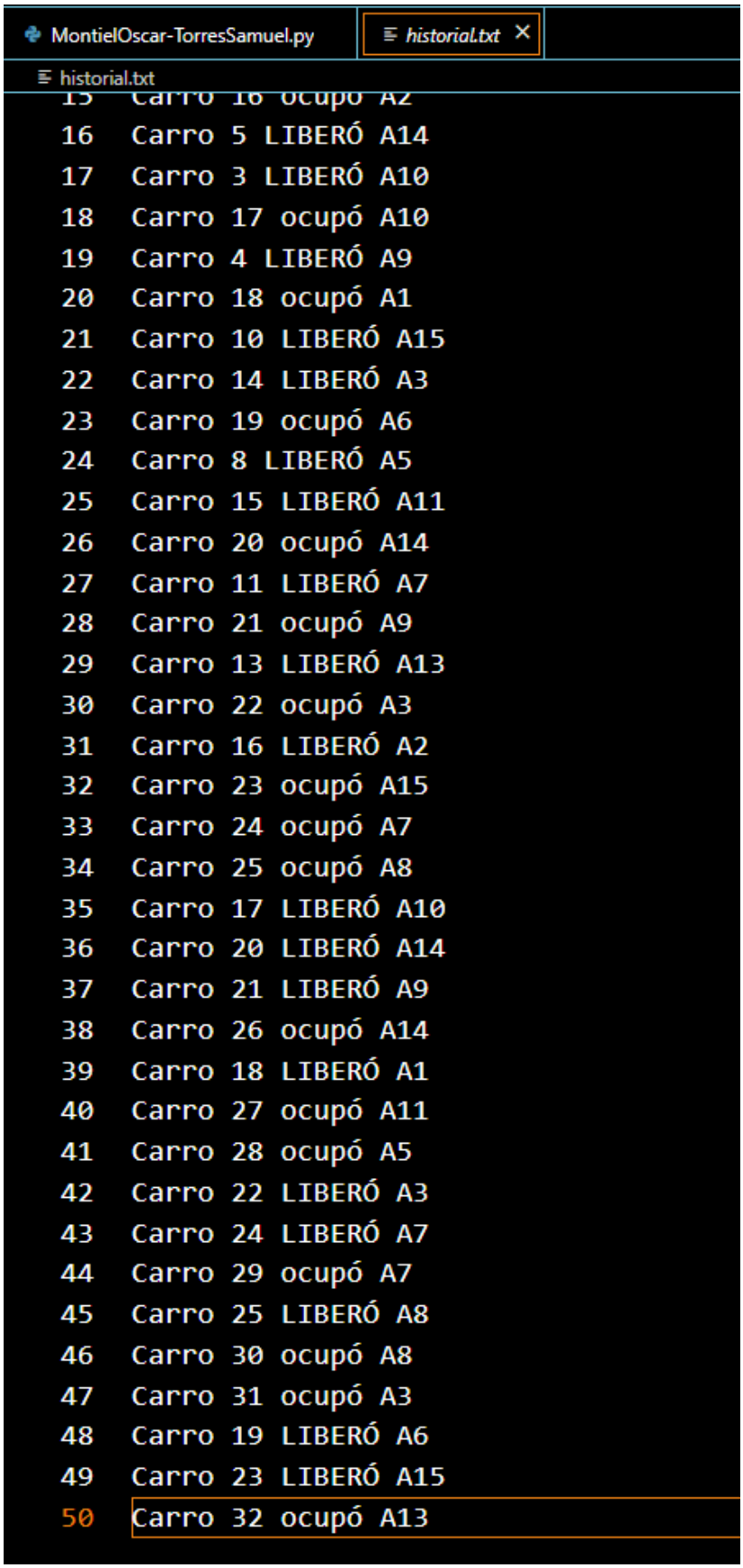
4. Salida esperada



Aquí se puede observar como el estacionamiento se llena poco a poco, en este caso quedan 7 lugares libres y 8 ocupados. La interfaz refleja **solo los espacios físicos (15)**, no el sobrecupo permitido (20)



En esta situación los lugares están totalmente ocupados por lo que los carros que están en el estacionamiento (el sobrecupo) buscarán estacionarse. Estos vehículos competirán por espacios liberados o serán retirados por la grúa si se estacionan mal


MontielOscar-TorresSamuel.py
historial.txt
historial.txt
15 Carro 16 ocupó A2
16 Carro 5 LIBERÓ A14
17 Carro 3 LIBERÓ A10
18 Carro 17 ocupó A10
19 Carro 4 LIBERÓ A9
20 Carro 18 ocupó A1
21 Carro 10 LIBERÓ A15
22 Carro 14 LIBERÓ A3
23 Carro 19 ocupó A6
24 Carro 8 LIBERÓ A5
25 Carro 15 LIBERÓ A11
26 Carro 20 ocupó A14
27 Carro 11 LIBERÓ A7
28 Carro 21 ocupó A9
29 Carro 13 LIBERÓ A13
30 Carro 22 ocupó A3
31 Carro 16 LIBERÓ A2
32 Carro 23 ocupó A15
33 Carro 24 ocupó A7
34 Carro 25 ocupó A8
35 Carro 17 LIBERÓ A10
36 Carro 20 LIBERÓ A14
37 Carro 21 LIBERÓ A9
38 Carro 26 ocupó A14
39 Carro 18 LIBERÓ A1
40 Carro 27 ocupó A11
41 Carro 28 ocupó A5
42 Carro 22 LIBERÓ A3
43 Carro 24 LIBERÓ A7
44 Carro 29 ocupó A7
45 Carro 25 LIBERÓ A8
46 Carro 30 ocupó A8
47 Carro 31 ocupó A3
48 Carro 19 LIBERÓ A6
49 Carro 23 LIBERÓ A15
50 Carro 32 ocupó A13

En este caso el archivo historial.txt guarda los últimos 50 eventos del sistema. (La tabla en consola muestra únicamente los 10 eventos más recientes).