TFI: Concurrencia

CINE M&L  
  
E.T. Nº 36  
ALMIRANTE GUILLERMO BROWN

  
Profesor: Alan Uzal  
Prog. Sobre Redes  
Trabajo Práctico - Concurrencia  
Año: 6º, División: 3º, Turno: Mañana

Integrantes:  
Godoy, Magalí  
Palombo, Lucía  
Fecha de entrega: 14/07/2025

Índice

1. Objetivo principal....................................................................................... 2

2. Introducción................................................................................................ 2

3. Arquitectura (Modelo en capas).................................................................. 2

4. Manejo de la concurrencia........................................................................... 2

5. Diagrama de clases....................................................................................... 3

6. Base de datos................................................................................................ 4

7. Diagrama de Gantt........................................................................................ 4

8. Conclusiones................................................................................................. 4

Objetivo principal:

Quisimos implementar nuestro propio cine, donde se puedan gestionar funciones como agregar y ver películas, elegir y agregar sala, comprar y ver entradas, registrar clientes, seleccionar horario de la función de dicha película. Es mucho más simple que cualquier otro cine. De esta forma se podrán estar realizando condiciones de carrera, interbloqueos, concurrencia.

Introducción:

Este trabajo consistió en el diseño e implementación de un sistema en Python para la gestión de un cine. La aplicación permite administrar películas, salas y entradas, con una estructura basada en el modelo de capas (dal, bll y gui). Además, se integró una base de datos SQLite para almacenar la información de forma persistente.

Una parte central de desarrollo fue el manejo de la concurrencia mediante el uso de hilos y procesos. Esto permitió simular múltiples compras de entradas realizadas al mismo tiempo por distintos clientes y ejecutar tareas paralelas como la exportación de la cartelera o el conteo de entradas. Estos mecanismos reflejan situaciones reales en sistemas productivos, donde múltiples operaciones ocurren de manera simultanea y deben ser controladas para evitar errores o perdidas de datos.

Arquitectura (Modelo en capas):  
Funcionamiento y lógica:  
Capa de presentación: Interfaz por consola (gui.py).  
Capa lógica: Servicios que manejan la lógica del negocio (bll.py).  
Capa de datos: Acceso y manipulación de la base de datos (dal.py).

Manejo de la concurrencia:

Durante el desarrollo enfrentamos varios desafíos relacionados con la concurrencia:

* Uno de los principales problemas fue la posibilidad de condiciones de carrera al comprar entradas. Por ejemplo, si dos clientes intentaban comprar entradas al mismo tiempo para la misma película, sin control de acceso, podría generarse una inconsistencia en el conteo o la capacidad de las salas.

Para resolver esto, implementamos hilos usando threading.Thread para representar múltiples clientes comprando entradas en simultaneo. Para evitar conflictos, utilizamos la clase Lock de threading, que nos permitió asegurar exclusión mutua en los tramos críticos del código, garantizando que solo un hilo pueda ejecutar cierta sección a la vez.

* Además, utilizamos procesos independientes como multiprocessing.Process para realizar tareas paralelas como:
  + exportar\_cartelera(): guarda la cartelera en un archivo de texto.
  + contar\_entradas(): calcula el total de las entradas vendidas.
  + enviar\_resumen(): simula el envio de un resumen diario de ventas.

Estas tareas se ejecutan de manera concurrente sin interferir entre sí ni con los hilos activos.

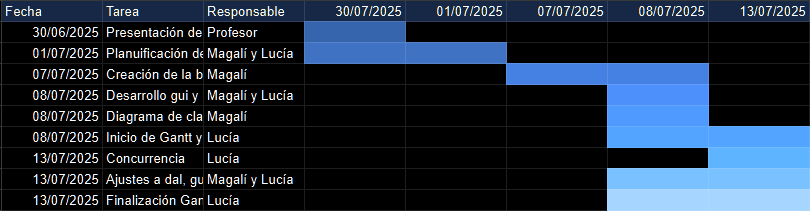
Otros posibles problemas de concurrencia y como se podrían solucionar son:

* Interbloqueo: se produce cuando dos o mas hilos quedan bloqueados esperando recursos que el otro posee y ninguno puede continuar. Por ejemplo: un hilo accede al archivo de cartelera y queda esperando acceso a la base de datos, mientras otro accedió a la base de datos y queda esperando el archivo. Para solucionar esto se establece un orden fijo para adquirir recursos o usar timeouts para liberar los recursos si no se obtiene acceso en un tiempo determinado.
* Starvation: ocurre cuando un hilo nunca accede al recurso compartido porque otros siempre tienen prioridad. Por ejemplo: un cliente con baja prioridad no logra comprar entradas porque siempre hay otros comprando antes. Para solucionar esto se implementa una política justa de acceso (por ejemplo, fifo: primero en llegar, primero en ser atendido) o usar prioridades que aumenten con el tiempo.

Principales clases y funciones relacionadas con la concurrencia:

- CompraThread: clase que define cada hilo de compra concurrente.  
- exportar\_cartelera(): Proceso que guarda la cartelera en un archivo de texto.  
- contar\_entradas(): proceso que calcula el total de entradas vendidas.  
- enviar\_resumen(): proceso que simula el envío del resumen de ventas.

Diagrama de clases:  


Diagrama de Gantt:  


[Diagrama de Gantt](https://docs.google.com/spreadsheets/d/1ZMrM4VLZ25cpdojiuEn90IXwZyYSFgQXEEsZz3TRiN8/edit?usp=sharing)

Base de datos:

La aplicación usa una base de datos local SQLite llamada cine\_M&L.db. Contiene 3 tablas principales:  
- Sala: almacena las salas del cine con su nombre y capacidad.  
- Peliculas: contiene las películas con su título, horario y referencia a la sala donde se proyectan.  
- Entradas: registra las compras de entradas, indicando al cliente, la cantidad y la película correspondiente.

Conclusiones:

Durante el desarrollo de este trabajo aprendimos a manejar la concurrencia en Python, implementando hilos y procesos para simular operaciones simultáneas y mejorar el rendimiento. Fue fundamental el uso de Lock para poder evitar condiciones de carrera y garantizar la integridad de los datos. Además, aplicamos conceptos de programación orientada a objetos y diseño en capas.