

1. DESCRIPCIÓN DEL PROYECTO

Este proyecto consiste en el desarrollo de un sistema distribuido y concurrente en Python que simula la gestión del tráfico aéreo. El sistema permite controlar múltiples aviones que solicitan aterrizaje o despegue, gestionando las pistas disponibles y visualizando en tiempo real el estado del sistema mediante un monitor.

Está compuesto por cuatro scripts principales que se ejecutan como procesos independientes y se comunican entre sí:

`iniciar_sistema.py`: se encarga de orquestar el sistema completo, lanzando el monitor, la torre de control y los aviones.

`torre.py`: es el componente central que gestiona las pistas, recibe las solicitudes de los aviones y envía actualizaciones al monitor.

`avion.py`: simula aviones individuales que solicitan aterrizaje o despegue.

`monitor.py`: muestra en tiempo real el estado del sistema, incluyendo las operaciones activas, las solicitudes pendientes y el historial de operaciones completadas.

2. QUÉ PROBLEMA RESUELVE

El sistema resuelve el problema clásico de la gestión concurrente de recursos limitados: en este caso, las pistas de aterrizaje y despegue. En un entorno real, varias aeronaves pueden solicitar acceso a un mismo recurso (una pista), y es necesario implementar mecanismos que aseguren la asignación justa, la espera ordenada y la correcta ejecución de cada operación sin conflictos ni condiciones de carrera.

Además, el sistema proporciona visualización clara de todo lo que ocurre y permite lanzar simulaciones automáticas o manejar vuelos de forma manual desde otra terminal, lo que lo hace flexible para distintos escenarios.

3. CÓMO ESTÁ ESTRUCTURADO

La arquitectura del sistema es modular y distribuida:

`iniciar_sistema.py`: gestiona el arranque, la interfaz del usuario y la simulación. Permite elegir entre modo automático (con múltiples aviones generados aleatoriamente) o modo manual. Además, muestra un resumen final al terminar cada simulación.

`torre.py`: mantiene la lógica principal. Gestiona una cola de solicitudes de vuelos, autoriza operaciones cuando hay pistas libres y actualiza al monitor. Usa exclusión mutua y semáforos para garantizar que solo un avión utilice una pista a la vez.

`avion.py`: representa cada avión como un proceso que envía una solicitud a la torre. Espera autorización, realiza su operación (aterrizaje o despegue) y finaliza.

`monitor.py`: recibe actualizaciones constantes de la torre mediante sockets TCP y actualiza la vista del sistema en la consola, además de guardar el historial de operaciones en un archivo JSON.

4. TÉCNICAS DE CONCURRENCIA, PARALELISMO Y DISTRIBUCIÓN UTILIZADAS

El sistema aplica varios conceptos clave:

Procesos independientes: Cada componente se ejecuta como un proceso separado utilizando `subprocess.Popen`, permitiendo verdadera concurrencia entre el monitor, la torre y los aviones.

Sockets TCP (socket): Se usa para la comunicación entre la torre de control y el monitor. Permite enviar actualizaciones de estado de forma asincrónica y distribuida.

Sincronización y exclusión mutua: Uso de semáforos, locks y estructuras de cola para evitar condiciones de carrera al asignar pistas y acceder a estructuras compartidas.

Archivos compartidos (.monitor_lock): Se usa como mecanismo simple de señalización para que el monitor sepa cuándo pausar la actualización de pantalla y permitir la impresión de resultados sin interferencias.

La elección de estas técnicas se basa en la necesidad de mantener procesos separados que se comuniquen de forma eficiente y sin bloqueo, permitiendo al mismo tiempo flexibilidad y control total sobre la ejecución de cada parte del sistema.

5. CAPTURAS O EJEMPLOS DE EJECUCIÓN

SISTEMA DE CONTROL AÉREO DISTRIBUIDO Y CONCURRENTE

Componentes del sistema:

1. Monitor de vuelos - Visualización en tiempo real
2. Torre de control - Gestión del tráfico aéreo
3. Simulación de tráfico aéreo - Modo automático o manual

Presione Ctrl+C en cualquier momento para detener todos los componentes

¿Desea iniciar una simulación automática?

1. Sí
2. No, usaré modo manual

Seleccione una opción (1-2):

=====

SISTEMA DE CONTROL AÉREO - MONITOR DE VUELOS
Última actualización: 19:13:36

=====

ESTADO DEL SISTEMA:

- Pistas disponibles: 1/2
 - Operaciones completadas: 0
 - Tiempo de espera promedio: 0.00s
-

OPERACIONES ACTIVAS:

ID VUELO	OPERACIÓN	PISTA	TIEMPO
UA4061	Aterrizaje	1	1.13s

SOLICITUDES PENDIENTES:

No hay solicitudes pendientes en este momento.

OPERACIONES RECIENTES:

No hay operaciones completadas aún.

19:13:37 - [Avión BA3102] Iniciando vuelo - Operación: despegue

19:13:37 - [Avión BA3102] Iniciando vuelo - Operación: despegue

Programando avión 3/10 - lanzamiento en 0.8s

19:13:37 - [Avión BA3102] Enviando solicitud de despegue a la torre

19:13:37 - [Avión BA3102] Enviando solicitud de despegue a la torre

[TORRE] Solicitud recibida: BA3102 quiere despegue

[TORRE] BA3102 comienza despegue en pista 1


19:13:37 - [Avión BA3102] Autorización recibida para despegue en pista 1 (espera: 0.00s)

19:13:37 - [Avión BA3102] Autorización recibida para despegue en pista 1 (espera: 0.00s)

19:13:37 - [Avión BA3102] Iniciando despegue en pista 1...

19:13:37 - [Avión BA3102] Iniciando despegue en pista 1...

=====

HISTORIAL DE VUELOS GESTIONADOS				
ID VUELO	OPERACIÓN	PISTA	TIEMPO	HORA
AA7468	despegue	1	2.51s	19:09:21
AA5934	despegue	1	2.50s	19:09:23
IB3312	aterrizaje	2	3.02s	19:09:25
BA9779	despegue	1	2.50s	19:09:26
AF9979	aterrizaje	2	3.01s	19:09:28
IB7947	aterrizaje	1	3.02s	19:09:29
IB5848	aterrizaje	2	3.00s	19:09:32
DL1135	despegue	1	2.51s	19:09:33
UA8488	aterrizaje	2	3.02s	19:09:36
LH2653	aterrizaje	1	3.01s	19:09:36
UA8985	despegue	2	2.51s	19:09:38
IB1236	despegue	1	2.52s	19:09:39
AA6446	despegue	2	2.52s	19:09:41
AA8269	aterrizaje	1	3.01s	19:09:42
IB6657	aterrizaje	2	3.02s	19:09:44
AF3680	despegue	1	2.50s	19:09:44
BA3995	aterrizaje	2	3.01s	19:09:47
 SIMULACIÓN COMPLETADA: 17 AVIONES GESTIONADOS				

5.1. ANÁLISIS DE RENDIMIENTO

Se comparó el comportamiento de una versión inicial más secuencial (con poca separación entre procesos) con la versión distribuida y concurrente actual. Los resultados mostraron:

Mejor capacidad de respuesta al simular múltiples vuelos al mismo tiempo.

Reducción del tiempo de espera gracias a la gestión simultánea de operaciones.

El sistema es más fluido y flexible cuando los componentes se ejecutan de forma independiente, especialmente en simulaciones grandes (más de 10 aviones).

6. INSTRUCCIONES DE EJECUCIÓN Y DEPENDENCIAS

Dependencias:

Python 3.9 o superior.

No se requieren paquetes externos: se usa solo la biblioteca estándar (asyncio, subprocess, socket, json, etc.).

Ejecución:

Colocar todos los archivos (monitor.py, torre.py, avion.py, iniciar_sistema.py) en la misma carpeta.

- Ejecutar el sistema con:
`python iniciar_sistema.py`
- Elegir el modo automático o manual.
- Para modo manual, se pueden lanzar aviones desde otra terminal con:
`python avion.py IB3456 (identificador del avión) aterrizaje`
`python avion.py AF7032 despegue`