# INFORME TÉCNICO PROYECTO FINAL INDIVIDUAL

## DESCRIPCIÓN DEL PROYECTO:

Este proyecto consiste en la creación de un simulador concurrente para un sistema de pedidos en línea, que incorpora conceptos clave de concurrencia, sincronización, exclusión mutua y visualización en tiempo real.

El sistema simula un entorno donde múltiples clientes realizan solicitudes de productos (como anillos, relojes, collares…) y varios repartidores se ocupan de las entregas. La dinámica sigue el modelo productor-consumidor, donde los productores son los clientes que realizan pedidos, y los consumidores son los repartidores que los recogen de una cola común y los entregan.

**La interfaz visual está realizada con la biblioteca Tkinter, es interactiva e intuitiva y presenta:**

→ La situación presente de cada repartidor (libre/ ocupado)

→ Un seguimiento en tiempo real de los acontecimientos del sistema (generación de pedidos, entregas, duración)

→ Un botón para comenzar la simulación.

→ Un resumen de estadísticas finales al terminar la ejecución.

**Las metas del proyecto son:**

→ Implementar y evidenciar métodos de programación concurrente en Python (threading, queue, block).

→ Ver de manera clara cómo se administran tareas simultáneas en un sistema realista.

→ Establecer una base que pueda ser reutilizada para futuras simulaciones en logística, educación o comercio.

Este tipo de simulación es útil en áreas como logística, comercio electrónico, servicios de entrega (food delivery) y resulta ser una herramienta educativa especialmente eficaz para ilustrar los principios de la concurrencia y la sincronización de procesos.

En cuanto a la experiencia del usuario, el simulador muestra visualmente los efectos de la concurrencia: se puede observar cómo los repartidores atienden pedidos en paralelo, cómo se alternan los estados de libre/ocupado, y cómo la eficiencia mejora cuando se permiten múltiples hilos ejecutándose en simultáneo.

## QUÉ PROBLEMA RESUELVE:

En diversos entornos como servicios de entrega a domicilio, comercios en línea, soluciones logísticas o sistemas de soporte al cliente, surgen ocasiones en las que numerosos usuarios efectúan solicitudes al mismo tiempo (pedidos, mensajes…), y un grupo restringido de empleados o procedimientos debe atenderlas. Este tipo de dificultad presenta problemas como:

→ La existencia de condiciones de carrera si múltiples procesos intentan acceder al mismo recurso sin la sincronización adecuada.

→ Ineficiencia si los empleados están descoordinados o están inactivos a la espera de tareas.

Este proyecto representa exactamente esa situación utilizando una arquitectura fundamentada en el patrón productor-consumidor. Los clientes realizan pedidos de forma aleatoria e independiente, mientras que los repartidores (recursos limitados) los recogen y entregan mediante una cola de tareas sincronizada.

DISTRIBUCIÓN DEL PROGRAMA:

1. Al iniciar el programa desde **main.py**, salta la ventana de la interfaz (**interfaz.py**).
2. Cuando se pulsa el botón "Iniciar Simulación":
    - Se crean varios hilos productores (clientes) que generan pedidos y los colocan en una cola.
    - Se lanzan varios hilos consumidores (repartidores) que recogen y entregan pedidos.
3. Los eventos serán mostrados en pantalla en tiempo real.
4. Al terminar todos los pedidos, el sistema calcula y muestra un resumen de rendimiento.

La estructura del proyecto permite modificar la cantidad de productores y consumidores, los tiempos de espera, o el comportamiento visual, sin alterar el núcleo del sistema.

```
simulador_pedidos/
├── main.py             # Punto de entrada del programa, lanza la interfaz gráfica
├── interfaz.py         # Interfaz de usuario con Tkinter, muestra estados y eventos
├── pedido.py           # Define la clase Pedido con sus atributos
├── productor.py        # Define la clase Productor (hilo que genera pedidos)
├── consumidor.py       # Define la clase Consumidor (hilo que procesa pedidos)
├── monitor.py          # Recoge estadísticas de reparto y calcula tiempos promedio
```

TÉCNICAS DE CONCURRENCIA Y PARALELISMO:

El proyecto emplea técnicas de concurrencia con hilos, siguiendo el patrón productor-consumidor, con mecanismos de sincronización que garantizan un buen acceso a los recursos compartidos.

**Concurrencia con hilos (threading.Thread)**

Cada cliente (productor) y repartidor (consumidor) es un hilo independiente. Esto permite que los pedidos se generen y se entreguen en paralelo, simulando un entorno real con varios actores actuando simultáneamente.

**Sincronización con queue.Queue**

Los pedidos generados se colocan en una cola compartida entre hilos. queue.Queue es segura para múltiples hilos.

**Exclusión mutua con threading.Lock**

El monitor de estadísticas (que registra tiempos y cantidades de entregas) se protege con un Lock, evitando condiciones de carrera al modificar datos compartidos.

**Patrón productor-consumidor**

Es el núcleo del sistema: los clientes generan pedidos y los repartidores los procesan desde una cola. Este patrón permite desacoplar la producción del consumo y distribuir eficientemente la carga.

**Separación de la interfaz gráfica**

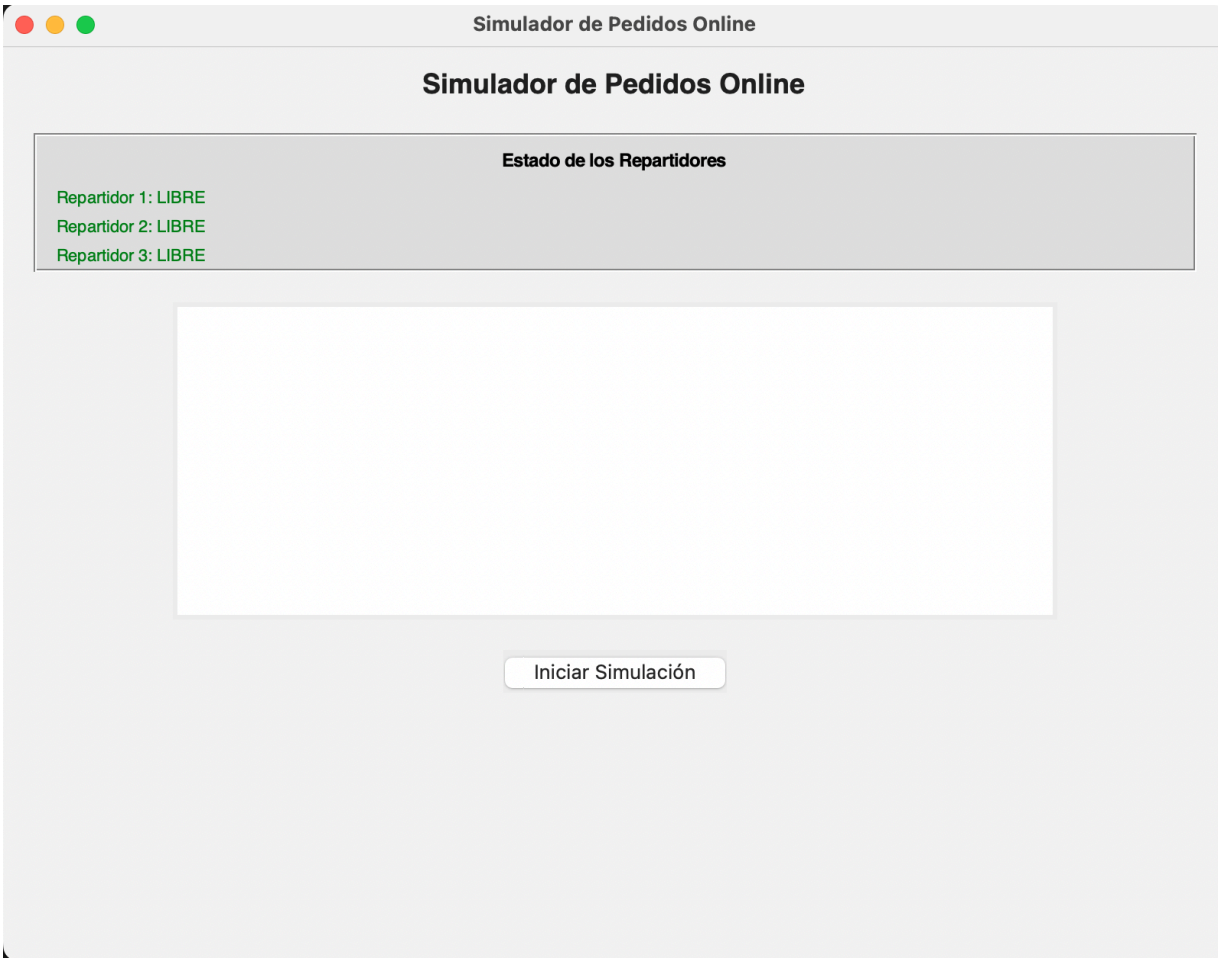
La simulación corre en un hilo aparte del bucle principal de Tkinter, mostrando actualizaciones en tiempo real sin bloquearse.

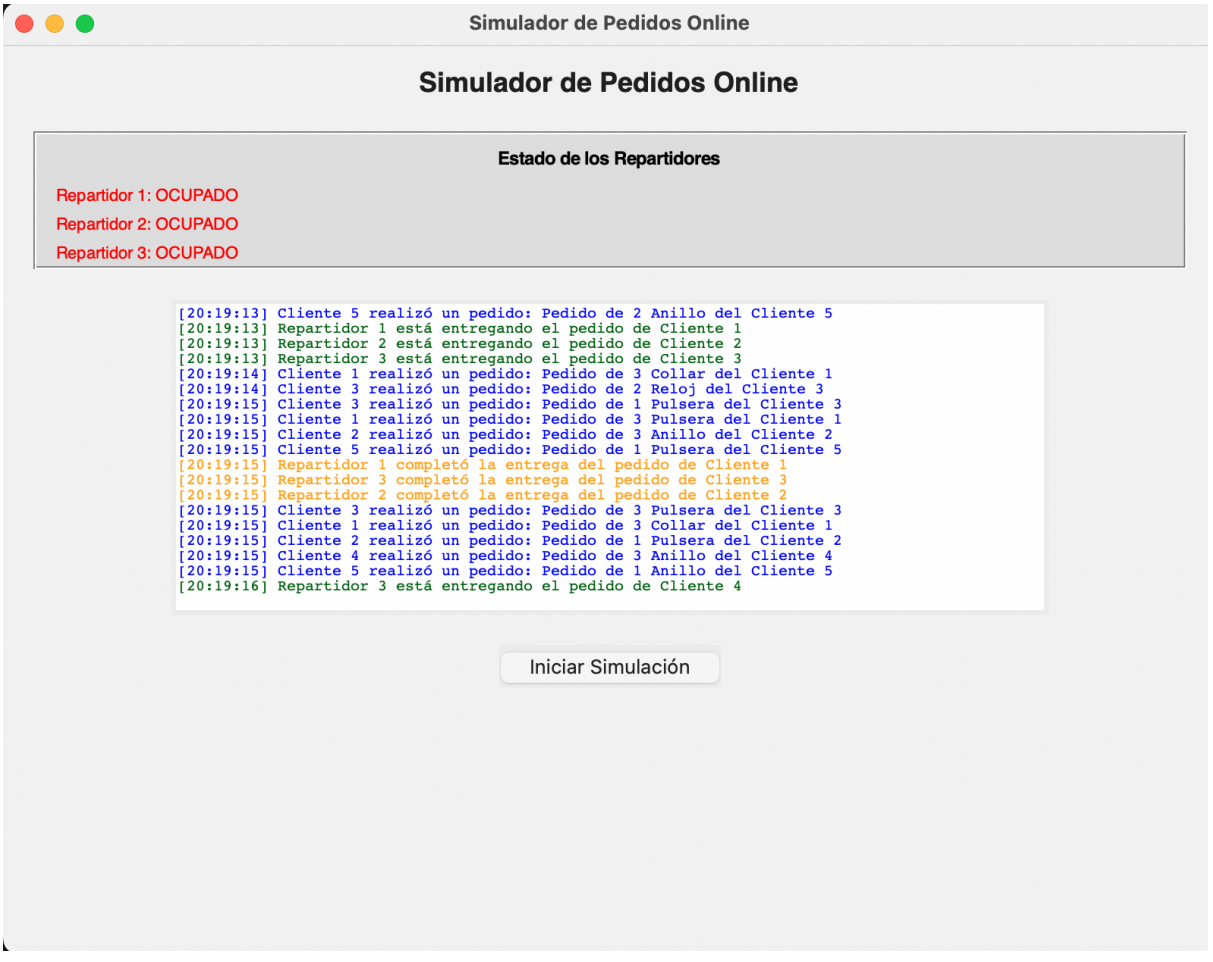
VERSIÓN SECUENCIAL VS VERSIÓN CONCURRENTE:

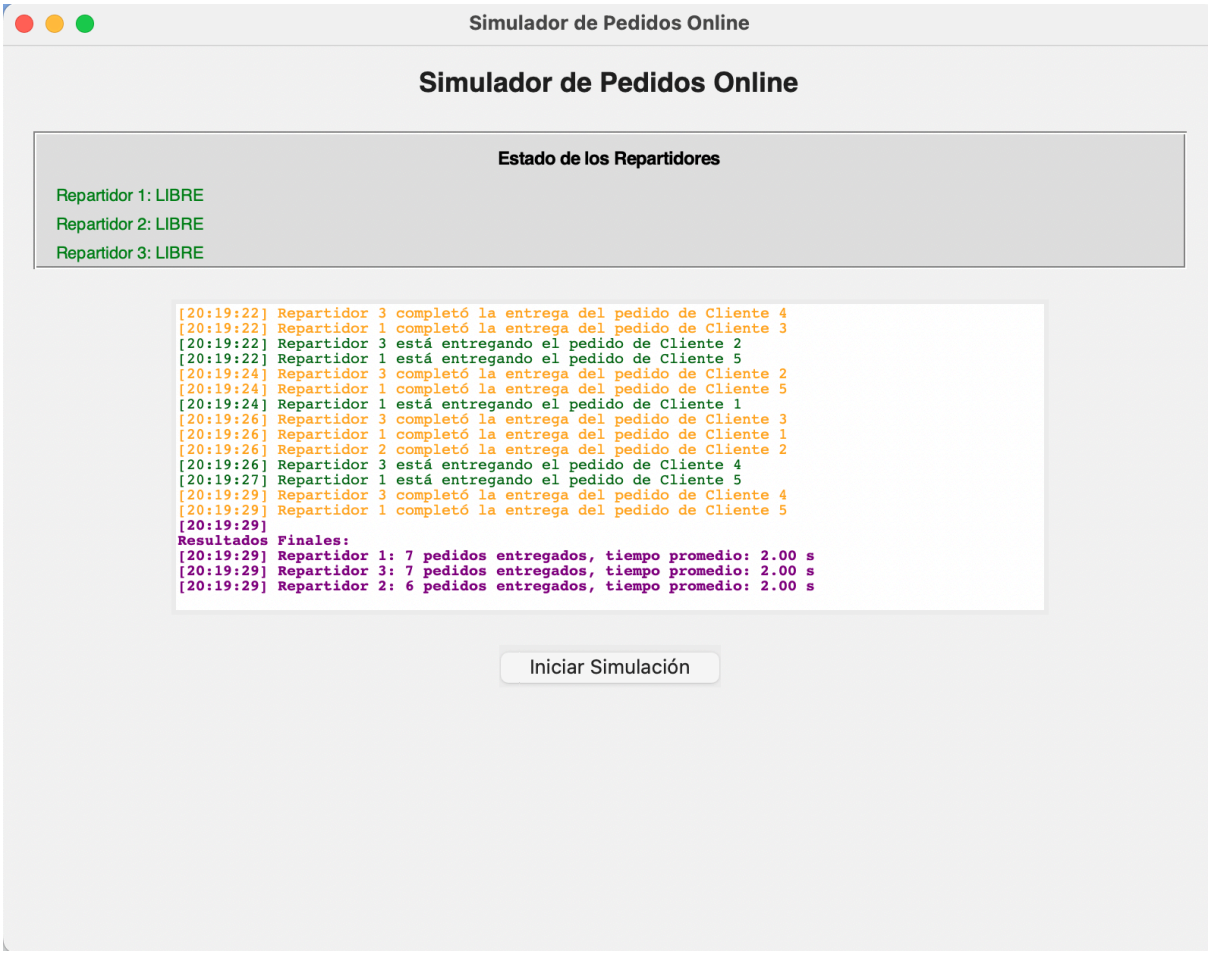
En la **versión secuencial**, los pedidos se gestionan uno tras otro. No existe simultaneidad en la producción y el reparto.

En la **versión concurrente**, varios clientes pueden generar pedidos al mismo tiempo, y varios repartidores los atienden en paralelo, lo que reduce el tiempo de respuesta.

El rendimiento mejora a medida que aumentan los hilos, aunque también puede llegar a saturarse si hay demasiada competencia por la cola o por el CPU.


Simulador de Pedidos Online
Simulador de Pedidos Online
Estado de los Repartidores
Repartidor 1: LIBRE
Repartidor 2: LIBRE
Repartidor 3: LIBRE
Iniciar Simulación


Simulador de Pedidos Online
Simulador de Pedidos Online
Estado de los Repartidores
Repartidor 1: OCUPADO
Repartidor 2: OCUPADO
Repartidor 3: OCUPADO
[20:19:13] Cliente 5 realizó un pedido: Pedido de 2 Anillo del Cliente 5
[20:19:13] Repartidor 1 está entregando el pedido de Cliente 1
[20:19:13] Repartidor 2 está entregando el pedido de Cliente 2
[20:19:13] Repartidor 3 está entregando el pedido de Cliente 3
[20:19:14] Cliente 1 realizó un pedido: Pedido de 3 Collar del Cliente 1
[20:19:14] Cliente 3 realizó un pedido: Pedido de 2 Reloj del Cliente 3
[20:19:15] Cliente 3 realizó un pedido: Pedido de 1 Pulsera del Cliente 3
[20:19:15] Cliente 1 realizó un pedido: Pedido de 3 Pulsera del Cliente 1
[20:19:15] Cliente 2 realizó un pedido: Pedido de 3 Anillo del Cliente 2
[20:19:15] Cliente 5 realizó un pedido: Pedido de 1 Pulsera del Cliente 5
[20:19:15] Repartidor 1 completó la entrega del pedido de Cliente 1
[20:19:15] Repartidor 3 completó la entrega del pedido de Cliente 3
[20:19:15] Repartidor 2 completó la entrega del pedido de Cliente 2
[20:19:15] Cliente 3 realizó un pedido: Pedido de 3 Pulsera del Cliente 3
[20:19:15] Cliente 1 realizó un pedido: Pedido de 3 Collar del Cliente 1
[20:19:15] Cliente 2 realizó un pedido: Pedido de 1 Pulsera del Cliente 2
[20:19:15] Cliente 4 realizó un pedido: Pedido de 3 Anillo del Cliente 4
[20:19:15] Cliente 5 realizó un pedido: Pedido de 1 Anillo del Cliente 5
[20:19:16] Repartidor 3 está entregando el pedido de Cliente 4
Iniciar Simulación




Estas son las tres etapas clave del funcionamiento del simulador. En la primera, los repartidores están en estado "libre" y comienzan a generarse los primeros pedidos (al pulsar el botón ‘Iniciar simulación’. En la segunda, los repartidores están activos y procesando entregas en paralelo, con actualizaciones en tiempo real en el registro de eventos. En la tercera etapa, todos los pedidos han sido terminados y se muestran los resultados finales con el resumen de entregas realizadas por cada repartidor y el tiempo promedio.