Sistema de Restaurante Automatizado

VÍCTOR VALDIVIA CALATRAVA

1.DESCRIPCIÓN

Como idea de proyecto, me he inspirado en uno de los primeros ejemplos que vimos con programación concurrente, los camareros. Basándome en el trabajo que se realiza en un restaurante, pero buscando la automatización y la optimización de la mayor cantidad de recursos posibles. El proyecto consta básicamente de una simulación de un restaurante replicando todo el entorno en un entorno programable con Python. La idea de esto es ver cómo se pueden automatizar los distintos procesos que ocurren en el restaurante, como la toma de órdenes al cliente, la recopilación de datos a nivel de ingresos, el tiempo que tardan los distintos procesos en llevarse a cabo e intentar automatizar estos procesos para optimizar y reducir al máximo los tiempos de espera

2.ESTRUCTURA

Para estructurar esta idea, he utilizado distintos scripts, cada uno representando una funcionalidad:

1. **host.py – Sala / Camarero:**
   Este script, simula principalmente los camareros, que se encargan de enseñar el menú a los distintos comensales y almacena estos pedidos en una cola, para que la cocina los procese posteriormente
2. **menu_api.py – Carta dinámica:**
   Utilizo este script para llamar a las APIs que contienen la carta de los platos y las bebidas. Utilizo ayncio y aiohttp para ejecutar ambas peticiones de forma paralela y así optimizar tiempos.
3. **kitchen.py – Cocina paralela:**
   Simula la cocina, con varios camareros, cada uno representado con un hilo para lo que utilizo threading. La tarea de cada hilo es extraer pedidos de la cola y simular el cocinado del plato correspondiente. Además, registro el tiempo individual de cada preparación y los ingresos con cada venta.
4. **performance.py – Métricas:**

Acumula el número de pedidos, el tiempo total de ejecución y los ingresos totales, haciendo un seguimiento de cada jornada para mostrarla al usuario al terminar la ejecución

5. **main.py – Orquestador:**
   Es el script principal desde el cual se ejecuta la simulación del restaurante, iniciando la toma de pedidos (utilizando asincronía), lanza el funcionamiento de la cocina con hilos y mide la duración global del preparado.

   El script de "settings.py" forma parte de la configuración interna del restaurante donde asigno el número de cocineros, las APIs en una variable

3.IMPLEMENTACIÓN Y RESOLUCIÓN DE PROBLEMAS REALES

En la restauración, hay muchos pequeños procesos que pueden alargar la toma de pedidos al cliente y la coordinación de todo el equipo, llegando a provocar largos tiempos de espera para el cliente, y no precisamente porque los pedidos requieran mucho tiempo de elaboración.

Con la automatización de las tareas que llevaría a cabo un camarero, el cliente ahorraría todo el tiempo que el camarero puede tardar en llegar a la mesa, traer la carta, todo esto mientas atiende a varias mesas y lleva los pedidos a cocina, pudiendo ahorrar e esto hasta 10/15 minutos, ya que recibe la carta instantáneamente nada más llegar, y los pedidos se almacenan en una cola que llegar directamente a cocina, evitando además errores que pueda tener el camarero al retener o apuntar la información, o incluso trasladándolo incorrectamente a los cocineros.

Aunque muchas de las implementaciones del programa son una simulación (los cocineros, por ejemplo) los procesos como la toma de órdenes, que he explicado anteriormente, e incluso, para los empleados todo el contador de tiempos de ejecución e ingresos, pueden suponer métricas clave para la mejora de los servicios que ofrecen.

4.TÉCNICAS UTILIZADAS

Para la optimización de los recursos utilizados, He utilizado dos conceptos clave vistos en clase:

el “paralelismo” que nos ofrecen los hilos con threading (concurrencia en Python por el GIL) para elaborar tareas sencillas y la asincronía para las llamadas a la APIs. Además, es importante el uso del lock para evitar que los hilos accedan de forma De forma desordenada a la cola donde se almacenan las distintas órdenes y evitar modificarlas y sacarlas a la vez, llegando a generar errores (condiciones de carrera).

La asincronía aparece en menu_api.py. Cada vez que un cliente solicita la carta, el programa necesita visitar dos servicios externos: uno que ofrece platos y otro que ofrece bebidas. Si esas peticiones se lanzasen de forma secuencial, la sala quedaría esperando a que termine la primera descarga para iniciar la segunda, pero con asyncio y aiohttp se realizan de manera simultánea. Mientras la red responde, el event-loop continúa, por lo que el usuario recibe la carta en prácticamente el mismo tiempo que tarda la petición más lenta. A nivel práctico, esto representa unos segundos ahorrados por mesa en la obtención del menú, lo que, en hora punta, se traduce en varios turnos de clientes adicionales a lo largo del día.

Una vez los pedidos están en la cola compartida, entra en juego el paralelismo real (salvo por ser Python) de kitchen.py. Aquí cada cocinero se materializa como un hilo independiente. Los hilos son ligeros y realizan tareas muy sencillas, haciéndolos perfectos para esta situación porque el “cocinado” no es una operación matemática pesada, sino una simulación de espera con temporizadores; mientras un hilo “duerme” (time.sleep), los demás pueden avanzar. El resultado es que siete cocineros virtuales (valor configurable en settings.py) preparan siete platos al mismo tiempo, de modo que la producción de la cocina escala casi de forma lineal.

Para que todo esto funcione sin conflictos, los datos compartidos se protegen con mecanismos mínimos pero efectivos. Con Queue ya se ofrece la garantía de seguridad entre hilos, así que camareros y cocineros pueden empujar o sacar pedidos sin bloquearse mutuamente. El único posible problema es la actualización del módulo de métricas, donde varios cocineros podrían querer incrementar simultáneamente el contador de ingresos; allí se cierra un lock apenas unos microsegundos, lo justo para sumar y seguir trabajando. Esta estrategia para controlar la sección crítica mantiene el programa seguro sin llegar a afectar el rendimiento o producir condiciones de carrera.

5.IMPLEMENTACIONES FUTURAS Y CONCLUSIÓN

Además, esto deja abierto muchas puertas para mejorar a futuro el programa. Por ejemplo, se podrían también manejar los costes de producción, es decir, el coste de los alimentos, el coste de la mano de obra (cocineros) Y así se pueden tener en cuenta tanto los ingresos como los costes para evaluar las ganancias. Además, se podrían sacar muchos datos acerca de los clientes para hacer recomendaciones cuando vuelvan a venir, y así también incitarles a acudir más al restaurante e incluso invitar a varios cercanos Haciendo crecer el número de clientes. En definitiva, con la optimización de los recursos y la automatización de los distintos procesos del restaurante se pueden obtener muchísimos beneficios tanto para los clientes como para el restaurante.