UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistemas Operativos.

Profesor: Ing. Gunnar Eyal Wolf Iszaevich.

Grupo: 06

Alumno: Hernández Hernández Samuel

Proyecto 02: Simulación de un restaurante.

Fecha de entrega: 06/11/2023

Identificación y descripción del problema

La administración de un restaurante durante el horario de servicio es un proceso complejo y dinámico que presenta características de concurrencia. En este escenario de la vida real, múltiples actores se entrelazan para proporcionar una experiencia gastronómica satisfactoria a los clientes, los protagonistas de este proceso incluyen a los comensales, los meseros, los cocineros y el personal de limpieza.

El proceso inicia con los comensales llegando al restaurante y tomando asiento en diferentes mesas, la concurrencia se hace evidente cuando los meseros deben tomar simultáneamente los pedidos de distintas mesas. Estos pedidos son transmitidos a la cocina, donde los cocineros preparan varios platos al mismo tiempo, coordinando sus esfuerzos para mantener el ritmo.

La sincronización es crucial, ya que los platos deben ser entregados a las mesas correctas en el momento adecuado, al mismo tiempo que deben coordinar con la cocina para garantizar que los platos estén listos.

La limpieza constante de las mesas es otra tarea concurrente, ya que, una vez que los comensales han terminado su comida, es importante que las mesas se limpien rápidamente para dar la bienvenida a nuevos clientes.

La resolución exitosa de este proceso requiere una gestión cuidadosa de la concurrencia, la coordinación entre los diferentes actores y una ejecución eficiente para garantizar que los comensales tengan una experiencia agradable y que todo funcione con ingenio y elegancia.

¿Dónde pueden verse las consecuencias nocivas de la concurrencia?

Retrasos en el servicio: Si no se gestiona adecuadamente, la concurrencia en la toma de pedidos y la preparación de alimentos puede resultar en retrasos en la entrega de platos a los comensales, esto puede llevar a una experiencia insatisfactoria para los clientes, que podrían tener que esperar mucho tiempo para recibir sus comidas.

¿Qué eventos pueden ocurrir que queramos controlar?

Confusión en la entrega: Cuando varios meseros entregan platos a las mesas al mismo tiempo, existe el riesgo de que se produzca confusión y se entreguen platos incorrectos a los comensales, esto puede generar insatisfacción y errores en los pedidos.

Congestión en la cocina: La cocina de un restaurante puede volverse caótica durante la cena si no se administra adecuadamente la concurrencia, los cocineros deben coordinar la preparación de varios platos simultáneamente, y si no se gestionan bien los tiempos y los recursos, puede haber congestión, retrasos y errores en la preparación de alimentos.

¿Hay eventos concurrentes para los cuales el ordenamiento relativo no resulta importante?

En cuanto a eventos concurrentes para los cuales el ordenamiento relativo no resulta importante, un ejemplo sería la solicitud simultánea de pedidos, si varios comensales en diferentes mesas piden pedidos al mismo tiempo, el orden en el que se sirvan estos puede no ser crítico, siempre y cuando todos los pedidos se entreguen de manera razonablemente rápida. En este caso, no se espera que los comensales se vean afectados por el orden en que se sirven los pedidos, siempre y cuando no haya un retraso excesivo.

Mecanismos de sincronización empleados

Se utilizan semáforos (threading.Semaphore) para gestionar el acceso a las mesas, limitar el número de cocineros trabajando y para coordinar la entrega de pedidos, los semáforos permiten controlar la concurrencia de los hilos y garantizar que no se excedan los recursos disponibles.

Lógica de operación

El programa simula la operación de un restaurante donde los comensales llegan, se sientan en una mesa y realizan un pedido a un mesero, los pedidos se envían a cocineros, que los preparan y luego se entregan a través de los meseros a los comensales, después de comer, los comensales pagan y se van, por último, el personal de limpieza limpia las mesas para los próximos comensales. La simulación continúa hasta que llegan 30 comensales, pero se puede ampliar ese número.

Identificación del estado compartido (variables o estructuras globales)

Las variables globales incluyen “message_queue” para la comunicación de eventos entre hilos, “comensales_llegados” para realizar un seguimiento del número de comensales y los semáforos mesas y cocineros para gestionar el acceso a las mesas y limitar el número de cocineros.

Descripción algorítmica del avance de cada hilo/proceso

- Los comensales llegan, se sientan en una mesa y realizan un pedido a un mesero.
- Los meseros toman pedidos de comensales en diferentes mesas sin esperar a otros meseros.
- Los cocineros preparan los pedidos y los entregan a través de los meseros.
- Los comensales comen y luego pagan y se van.
- El personal de limpieza limpia las mesas.
- La simulación termina después de que han llegado 30 comensales.

Descripción de la interacción entre ellos (sea mediante los mecanismos de sincronización o de alguna otra manera)

- Los comensales interactúan con los meseros al realizar pedidos.
- Los meseros interactúan con los comensales al tomar pedidos y entregar pedidos.
- Los cocineros interactúan con los meseros al entregar pedidos.
- El personal de limpieza interactúa con las mesas al limpiarlas.

Descripción del entorno de desarrollo, suficiente para reproducir una ejecución exitosa

¿Qué lenguaje emplean? ¿Qué versión?

El lenguaje de programación utilizado es Python versión 3.12

¿Qué bibliotecas más allá del estándar del lenguaje?

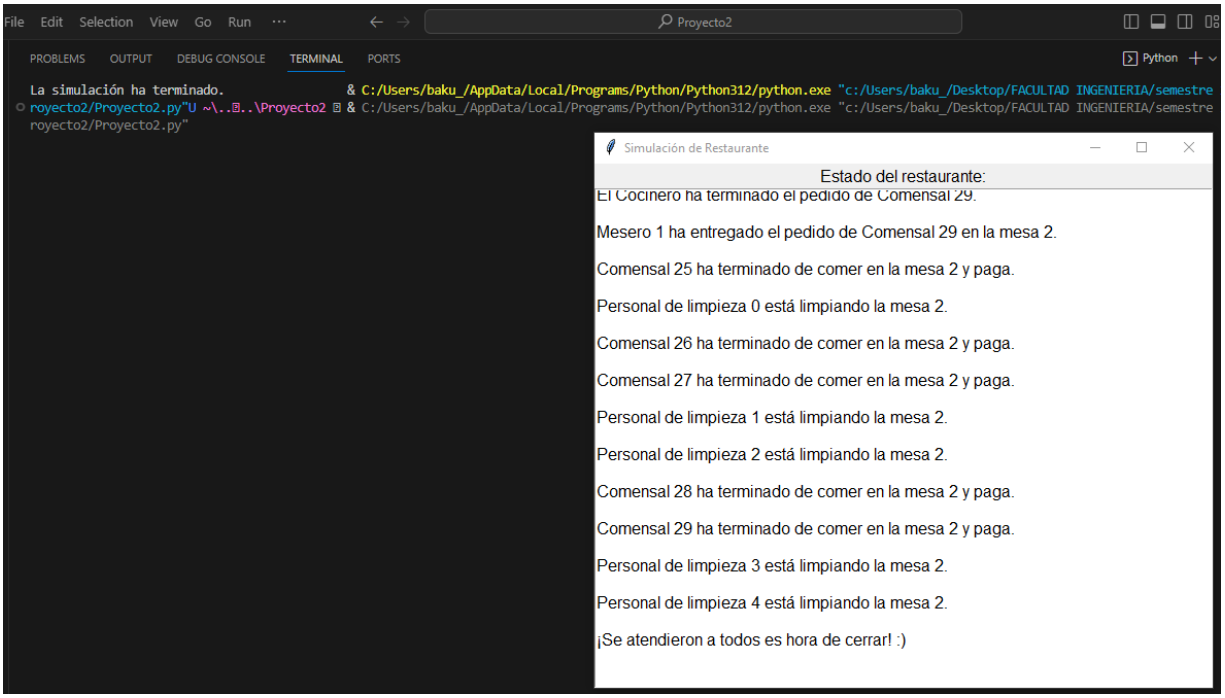
Se utiliza la biblioteca estándar “threading” para la gestión de hilos y “tkinter” para la interfaz gráfica.

¿Bajo qué sistema operativo / distribución lo desarrollaron y/o probaron?

El sistema operativo fue Windows 10, y se probo en el editor de texto Visual Studio Code

Para ejecutar este programa en Linux basta con abrir la consola con la dirección hasta donde esta el archivo “.py” y escribir python3 nombredelarchivo.py y dar enter

Ejemplos o pantallazos de una ejecución exitosa



Una vez que se cierra la interfaz grafica se muestra el mensaje “La simulación ha terminado” en la terminal.