Práctica 10: Productor - Consumidor

Maestro:

Javier Rosales Martinez

Materia:

Seminario de Solución de Problemas de Sistemas Operativos

Sección:

D06

Alumno:

Alejandro Covarrubias Sánchez

Código:

221350192

Antecedentes

Es un problema clásico en la programación concurrente que aborda los desafíos de sincronización entre dos procesos: un productor, que genera productos y los coloca en un búfer compartido, y un consumidor, que toma esos productos para su procesamiento. La coordinación entre ambos es esencial, ya que el productor no debe exceder la capacidad del búfer y el consumidor solo puede retirar productos si hay disponibles.

Una de las reglas fundamentales es que el búfer tiene una capacidad finita, lo que significa que el productor debe esperar si está lleno y el consumidor si está vacío. Para manejar esta sincronización, se utilizan semáforos, que ayudan a coordinar el acceso al búfer y evitan condiciones de carrera.

Metodología

Este código implementa una simulación de un estacionamiento utilizando multithreading y una interfaz gráfica en Python. La simulación incluye clases y métodos para gestionar la entrada y salida de autos, sincronizando las acciones concurrentes mediante hilos. La clase `Estacionamiento` encapsula la lógica principal, permitiendo añadir o retirar autos de manera sincronizada con el uso de `Lock` para evitar condiciones de carrera. Además, integra un sistema para modificar dinámicamente las frecuencias de entrada y salida mediante valores aleatorios.

La interfaz gráfica, creada con `tkinter`, permite visualizar en tiempo real el estado del estacionamiento, representando los autos como rectángulos en un lienzo (`canvas`). Cada acción de entrada o salida se refleja gráficamente, brindando una experiencia interactiva. El diseño es modular, con funciones específicas para manejar la representación gráfica y los ajustes dinámicos.

El programa utiliza multithreading para ejecutar simultáneamente la entrada de autos, la salida, y la modificación aleatoria de frecuencias, garantizando un funcionamiento fluido y realista.

```python
def agregar_auto(self):
    while True:
        with self.lock:
            if len(self.autos) < self.capacidad:
                nuevo_auto = Auto(len(self.autos) + 1)
                self.autos.append(nuevo_auto)
                self.dibujar_auto(len(self.autos) - 1)  # Dibujar auto en el canvas
                print(f"Entrada: Auto {nuevo_auto.id} añadido.")
            else:
                print("El estacionamiento está lleno. No se puede añadir más autos.")
        time.sleep(self.frecuencia_entrada)

def retirar_auto(self):
    while True:
        with self.lock:
            if len(self.autos) > 0:
                auto_retirado = self.autos.pop(0)
                self.borrar_auto()  # Borrar auto del canvas
                print(f"Salida: Auto {auto_retirado.id} retirado.")
            else:
                print("El estacionamiento está vacío. No se pueden retirar autos.")
        time.sleep(self.frecuencia_salida)
```

Conclusión

El código entregado implementa una simulación básica y funcional de un estacionamiento. Permite añadir y retirar autos en tiempo real mientras ajusta aleatoriamente las frecuencias de estas operaciones. Esto se visualiza en una interfaz gráfica desarrollada con `tkinter`, donde los autos son representados por rectángulos en un lienzo. La simulación también incluye mensajes en la consola para informar sobre las operaciones realizadas, como el ingreso o salida de autos y los cambios en las frecuencias.

Entre las funcionalidades ya implementadas y funcionales destacan el manejo sincronizado de los hilos para la entrada y salida de autos, el ajuste aleatorio de frecuencias de operación, y la representación visual en la interfaz gráfica. Además, el uso de `Lock` asegura que las modificaciones a la lista de autos sean seguras, evitando errores concurrentes. La interfaz es sencilla pero suficiente para representar el estado del estacionamiento de manera clara y en tiempo real.

En futuras versiones se podrían mejorar varios aspectos. Por ejemplo, la interfaz gráfica podría ampliarse con información adicional, como contadores de autos actuales o un indicador de estado del estacionamiento (lleno o vacío). También sería útil incluir controles interactivos para que el usuario ajuste manualmente las frecuencias de entrada y salida o para pausar la simulación. Desde un punto de vista técnico, la simulación podría beneficiarse de un manejo más robusto de errores, como qué hacer si los hilos enfrentan problemas inesperados.

Referencias

Academia Cimne Iber. (n.d.). *Productor-Consumidor*. Programación Avanzada. Academia Cimne Iber. https://progavanzada.academiacimneiber.com/leccion-8/02-producer-consumer/