Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Ingeniería en computación

Seminario de solución de problemas de sistemas operativos

NRC 164138

Sección D07

Profesor Javier Rosales Martínez

Diego Armando Sánchez Rubio 217570609

Reporte Práctica 10.

Antecedentes.

Esta práctica simula un estacionamiento con capacidad limitada, en el que entran y salen autos automáticamente. Cada auto se representa como un objeto, y el estacionamiento es una lista que contiene los autos actualmente dentro.

La entrada y salida de autos se controla mediante dos hilos que se ejecutan al mismo tiempo, con diferentes frecuencias configurables.

Metodología.

Lo primero que se hace en esta práctica es que se importan las librerías necesarias:

tkinter para la interfaz gráfica.

threading para el uso de hilos.

time para usar sleep() entre entradas/salidas.

random para seleccionar frecuencias aleatorias al inicio.

```
import tkinter as tk
import threading
import time
import random
```

Después de esto se definen variables globales que se utilizarán en el funcionamiento del programa:
tam_max: capacidad máxima del estacionamiento.

estacionamiento: lista que contiene los autos presentes.

frecuencia_entrada y frecuencia_salida: cada una toma un valor aleatorio (0.5, 1 o 2 segundos).

ejecutando: controla si el sistema sigue activo.

lock: un bloqueo para asegurar que no haya conflicto cuando varios hilos modifican la lista.

```
# --- Variables globales ---
tam_max = 12
estacionamiento = []
frecuencia_entrada = random.choice([0.5, 1, 2])
frecuencia_salida = random.choice([0.5, 1, 2])
ejecutando = True
lock = threading.Lock()
```

Lo siguiente es definir un clase Auto que se encarga de qué cada auto nuevo reciba un identificador único (A1, A2, etc.). Esto facilita su seguimiento y visualización en pantalla.

```python
# --- Clase Auto ---
class Auto:
    contador = 1
    Tabnine | Edit | Test | Explain | Document
    def __init__(self):
        self.id = Auto.contador
        Auto.contador += 1
    Tabnine | Edit | Test | Explain | Document
    def __str__(self):
        return f"A{self.id}"
```

Después necesitamos varias funciones, la primera es entrada_autos() que lo que hace es que es un hilo que se ejecuta continuamente y hace lo siguiente

- Espera el tiempo definido por frecuencia_entrada.
- Si hay espacio, agrega un nuevo auto al estacionamiento.
- Usa lock para evitar que otro hilo modifique la lista al mismo tiempo.
- Actualiza la GUI con el nuevo estado.

```python
def entrada_autos():
    global frecuencia_entrada
    while ejecutando:
        time.sleep(frecuencia_entrada)
        with lock:
            if len(estacionamiento) < tam_max:
                auto = Auto()
                estacionamiento.append(auto)
        actualizar_gui()
```

Después tenemos esta otra función que también corre en un hilo separado:

- Espera el tiempo de frecuencia_salida.
- Si hay autos, saca el primero (simula salida FIFO).
- También usa lock y actualiza la GUI.

```python
Tabnine | Edit | Test | Explain | Document
def salida_autos():
    global frecuencia_salida
    while ejecutando:
        time.sleep(frecuencia_salida)
        with lock:
            if estacionamiento:
                estacionamiento.pop(0)
        actualizar_gui()
```

La siguiente función se encarga de actualizar la interfaz:

- Si una celda está ocupada, muestra el ID del auto y la pinta de verde.
- Si está libre, la deja en blanco.

- También actualiza las etiquetas informativas.

```python
# --- Actualizar interfaz ---
Tabnine | Edit | Test | Explain | Document
def actualizar_gui():
    for i in range(tam_max):
        if i < len(estacionamiento):
            celdas[i].config(text=str(estacionamiento[i]), bg="lightgreen")
        else:
            celdas[i].config(text="", bg="white")
    estado_label.config(text=f"Ocupados: {len(estacionamiento)}/{tam_max}")
    entrada_label.config(text=f"Frecuencia Entrada: {frecuencia_entrada}s")
    salida_label.config(text=f"Frecuencia Salida: {frecuencia_salida}s")
```

Y por ultimo dentro de las funciones más importantes tenemos la que se encarga de cambiar al frecuencia dinámicamente.

- Se toma el valor del Entry (campo de texto).
- Se asigna a la frecuencia correspondiente si es válido.

```python
# --- Cambiar frecuencias ---
Tabnine | Edit | Test | Explain | Document
def cambiar_frecuencia(tipo):
    global frecuencia_entrada, frecuencia_salida
    try:
        valor = float(frecuencia_entry.get())
        if tipo == 'e':
            frecuencia_entrada = valor
        elif tipo == 's':
            frecuencia_salida = valor
        actualizar_gui()
    except ValueError:
        pass  # Silencio si el usuario escribe mal
```

Conclusiones.

Esta práctica si fue un poco más complicada que la anterior, ya que aunque se rige de varios conceptos de la pasada, tiene una lógica un poco más complicada al momento de manejar los hilos, pero aún así se obtuvo el resultado deseado.

Referencias.

- Python Software Foundation. (2023). *threading — Thread-based parallelism*. En *Python 3.11.5 documentation*. https://docs.python.org/3/library/threading.html

- Python Software Foundation. (2023). *Tkinter — Python interface to Tcl/Tk*. En *Python 3.11.5 documentation*. https://docs.python.org/3/library/tkinter.html

- Lundh, F. (2001). *An Introduction to Tkinter*. PythonWare. Recuperado de http://effbot.org/tkinterbook/

- Sweigart, A. (2019). *Automate the Boring Stuff with Python: Practical Programming for Total Beginners* (2nd ed.). No Starch Press.
[Capítulos sobre hilos e interfaces gráficas]

- Beazley, D. M. (2009). *Python Essential Reference* (4th ed.). Addison-Wesley.