Laboratorio del 3 corte

Autor: David Díaz, Jhonatan Vargas, David Lopez
Fecha: Noviembre 2025
Materia: Digitales 3

Objetivo General

Desarrollar aplicaciones en Python que utilicen procesamiento paralelo mediante **hilos** (`threading`) y **mutex** para sincronización, implementando proyectos de análisis de sentimientos, juegos 2D y detección de gestos de mano en tiempo real. Cada aplicación debe contar con una interfaz interactiva (Streamlit o Pygame) y ser portable mediante la creación de contenedores Docker, garantizando su ejecución segura y consistente en cualquier entorno.

Proyecto 1: Análisis de Sentimientos con Hilos

Descripción del Proyecto

El programa permite cargar un archivo de texto con múltiples comentarios, clasificándolos según su **sentimiento**: positivo, negativo o neutro. El procesamiento se realiza en paralelo mediante hilos, utilizando un `Lock` para proteger el acceso a la lista compartida de resultados.

Estructura del Proyecto

- **app.py**: interfaz de usuario con Streamlit.
- **sentiment.py**: lógica del análisis de sentimientos con hilos.
- **requirements.txt**: dependencias de Python.
- **Dockerfile**: instrucciones para construir la imagen de Docker.

Código Fuente Principal

sentiment.py

```
import threading

lock = threading.Lock()

positive_words = ["bueno", "excelente", "feliz", "incre ble", "me␣
    gusta", "fant stico"]
negative_words = ["malo", "horrible", "triste", "terrible", "odio", "
    p simo"]

def analyze_sentiments(comments):
    results = []
    threads = []

```

```
    def worker(subset):
        local_results = []
        for text in subset:
            text = text.lower()
            score = 0
            for p in positive_words:
                if p in text:
                    score += 1
            for n in negative_words:
                if n in text:
                    score -= 1

            if score > 0:
                sentiment = "Positivo"
            elif score < 0:
                sentiment = "Negativo"
            else:
                sentiment = "Neutro"

            local_results.append((text, sentiment))

        with lock:
            results.extend(local_results)

    chunk_size = max(1, len(comments)//4)
    for i in range(0, len(comments), chunk_size):
        subset = comments[i:i+chunk_size]
        t = threading.Thread(target=worker, args=(subset,))
        threads.append(t)
        t.start()

    for t in threads:
        t.join()

    return results
```

app.py

```
import streamlit as st
from sentiment import analyze_sentiments

st.title("An lisis de Sentimientos con Hilos en Paralelo")
st.write("Sube un archivo .txt con comentarios (uno por l nea) para
    analizar su sentimiento.")

uploaded = st.file_uploader("Selecciona un archivo", type=["txt"])

if uploaded:
    comments = [l.strip() for l in uploaded.read().decode('utf-8').
       splitlines() if l.strip()]
    if comments:
        st.write(f"Se cargaron {len(comments)} comentarios.")
        results = analyze_sentiments(comments)
        for c, s in results:
            st.write(f"**{s}:** {c}")
```

requirements.txt

```
streamlit
```

Dockerfile

```
FROM python:3.12-slim

WORKDIR /app
COPY . .

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8501

CMD ["streamlit", "run", "app.py", "--server.address=0.0.0.0"]
```

Ejecución del Proyecto

Modo local (sin Docker)

1. Crear entorno virtual:

   `python3 -m venv venv source venv/bin/activate pip install -r requirements.txt`

2. Ejecutar:

   `streamlit run app.py`

3. Abrir en el navegador: `http://localhost:8501`

Modo Docker

1. Construir imagen:

   `docker build -t sentiment-app .`

2. Ejecutar contenedor:

   `docker run -p 8501:8501 sentiment-app`

3. Abrir navegador en `http://localhost:8501`

Resultados Esperados

Al cargar un archivo con frases como:

`Me gusta este producto Odio cómo funciona Es aceptable`

La aplicación mostrará:

`Positivo: Me gusta este producto Negativo: Odio cómo funciona Neutro: Es aceptable`

Proyecto 2: Juego 2D Tipo Mario Bros con Hilos

Descripción del Proyecto

Implementa un juego de plataformas 2D donde el jugador puede moverse, saltar varias veces en el aire y recolectar monedas mientras evita enemigos. Cada elemento dinámico del juego es controlado mediante hilos y sincronización con `Lock`, evitando condiciones de carrera.

Estructura de Archivos

- `game.py`: código principal del juego.
- `requirements.txt`: dependencias (`pygame`).
- `Dockerfile`: imagen Docker con Python y Pygame.
- `assets/`: recursos opcionales (sprites, sonidos).

Funcionamiento

- Inicializa Pygame, crea la ventana, plataformas, jugador, enemigos y monedas.
- Jugador con movimiento horizontal y multi-salto.
- Enemigos y monedas gestionados mediante hilos.
- Uso de `Lock` para sincronizar posiciones y colisiones.
- Loop principal dibuja todos los elementos y actualiza puntuación y vidas.
- Fin de juego cuando el jugador pierde todas las vidas o cae.

Código Principal (game.py)

```
1  import pygame
2  import threading
3  from threading import Lock
4
5  pygame.init()
6  WIDTH, HEIGHT = 800, 400
7  WIN = pygame.display.set_mode((WIDTH, HEIGHT))
8  pygame.display.set_caption("Mario 2D con Hilos")
9
10 RED = (255, 0, 0)
11 GREEN = (0, 255, 0)
12 BROWN = (139,69,19)
13 YELLOW = (255, 255, 0)
14 WHITE = (255,255,255)
15
16 platforms = [(0, HEIGHT-50, WIDTH, 50), (100, 300, 200, 20), (400,
       200, 150, 20)]
17 lock = Lock()
18
```

```
class Player:
    def __init__(self):
        self.rect = pygame.Rect(50, HEIGHT-100, 40, 40)
        self.vel_y = 0
        self.jumping = 0
        self.lives = 3
        self.score = 0

    def move(self, keys):
        if keys[pygame.K_LEFT]:
            self.rect.x -= 5
        if keys[pygame.K_RIGHT]:
            self.rect.x += 5
        if keys[pygame.K_SPACE] and self.jumping < 2:
            self.vel_y = -10
            self.jumping += 1

    def apply_gravity(self):
        self.vel_y += 0.5
        self.rect.y += self.vel_y
        if self.rect.bottom > HEIGHT:
            self.rect.bottom = HEIGHT
            self.vel_y = 0
            self.jumping = 0

class Enemy(threading.Thread):
    def __init__(self, x, y):
        threading.Thread.__init__(self)
        self.rect = pygame.Rect(x, y, 40, 40)
        self.direction = 1

    def run(self):
        while True:
            with lock:
                self.rect.x += self.direction*2
                if self.rect.left < 0 or self.rect.right > WIDTH:
                    self.direction *= -1
            pygame.time.delay(50)

class Coin(threading.Thread):
    def __init__(self, x, y, player):
        threading.Thread.__init__(self)
        self.rect = pygame.Rect(x, y, 20, 20)
        self.player = player

    def run(self):
        while True:
            with lock:
                if self.rect.colliderect(self.player.rect):
                    self.player.score += 10
                    self.rect.x, self.rect.y = -100, -100
            pygame.time.delay(50)

player = Player()
enemies = [Enemy(300, HEIGHT-90), Enemy(500, HEIGHT-90)]
coins = [Coin(120, 270, player), Coin(450, 170, player)]

for e in enemies: e.start()
```

```
for c in coins: c.start()

clock = pygame.time.Clock()
run = True
while run:
    clock.tick(60)
    WIN.fill(WHITE)
    keys = pygame.key.get_pressed()
    player.move(keys)
    player.apply_gravity()
    for plat in platforms:
        pygame.draw.rect(WIN, BROWN, plat)
    pygame.draw.rect(WIN, RED, player.rect)
    for e in enemies:
        pygame.draw.rect(WIN, GREEN, e.rect)
    for c in coins:
        pygame.draw.rect(WIN, YELLOW, c.rect)
    pygame.display.flip()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
pygame.quit()
```

Proyecto 3: Detección de Gestos de Mano en Tiempo Real

Descripción del Proyecto

Sistema de detección de gestos de mano usando MediaPipe y Python, con procesamiento concurrente mediante hilos y protección de variables compartidas con mutex. La interfaz se implementa con Streamlit y se empaqueta en Docker.

Código Principal

```
import cv2
import mediapipe as mp
import streamlit as st
import threading
import time

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

gesture_frame = None
gesture_text = ""
frame_lock = threading.Lock()

def classify_gesture(hand_landmarks):
    tips = [hand_landmarks.landmark[i] for i in [
        mp_hands.HandLandmark.THUMB_TIP,
        mp_hands.HandLandmark.INDEX_FINGER_TIP,
        mp_hands.HandLandmark.MIDDLE_FINGER_TIP,
        mp_hands.HandLandmark.RING_FINGER_TIP,
```

```
        mp_hands.HandLandmark.PINKY_TIP]]
    mcp = [hand_landmarks.landmark[i] for i in [
        mp_hands.HandLandmark.THUMB_MCP,
        mp_hands.HandLandmark.INDEX_FINGER_MCP,
        mp_hands.HandLandmark.MIDDLE_FINGER_MCP,
        mp_hands.HandLandmark.RING_FINGER_MCP,
        mp_hands.HandLandmark.PINKY_MCP]]
    fingers_up = sum([tips[i].y < mcp[i].y for i in range(5)])
    if fingers_up == 5:
        return "Palma␣abierta"
    elif (tips[1].y < mcp[1].y and tips[0].y < mcp[0].y and fingers_up
        == 2):
        return "Pulgar␣e␣ ndice ␣en␣L"
    elif (tips[1].y < mcp[1].y and fingers_up == 1):
        return " ndice ␣levantado"
    else:
        return "Otro␣gesto"

def process_camera():
    global gesture_frame, gesture_text
    cap = cv2.VideoCapture(0)
    with mp_hands.Hands(min_detection_confidence=0.7,
                        min_tracking_confidence=0.5) as hands:
        while True:
            ret, frame = cap.read()
            if not ret:
                continue
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = hands.process(frame_rgb)
            text_overlay = []
            if results.multi_hand_landmarks:
                for i, hand_landmarks in enumerate(results.
                   multi_hand_landmarks):
                    mp_drawing.draw_landmarks(frame, hand_landmarks,
                       mp_hands.HAND_CONNECTIONS)
                    gesture = classify_gesture(hand_landmarks)
                    text_overlay.append(f"Mano␣{i+1}:␣{gesture}")
            with frame_lock:
                gesture_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                gesture_text = "\n".join(text_overlay)

st.title("Detecci n␣de␣Gestos␣de␣Mano")
st.text("Con␣enfoque␣en␣aprender␣e␣implementar␣en␣el␣segundo␣semestre␣
   2023")
st.write("https://github.com/SebastianPinzonC/Operativos-2023")

threading.Thread(target=process_camera, daemon=True).start()

frame_placeholder = st.empty()
gesture_placeholder = st.empty()
while True:
    with frame_lock:
        if gesture_frame is not None:
            frame_placeholder.image(gesture_frame)
            gesture_placeholder.text(gesture_text)
    time.sleep(0.03)
```

Conclusiones Generales

- El uso de hilos y mutex permite desarrollar aplicaciones concurrentes que procesan datos en paralelo de manera segura.
- Streamlit y Pygame facilitan la creación de interfaces interactivas sin requerir desarrollo web avanzado.
- Docker asegura la portabilidad y reproducibilidad de las aplicaciones en cualquier entorno.
- Estos proyectos pueden servir como base para futuras aplicaciones de visión por computadora, análisis de datos y juegos interactivos.

Referencias

- `https://github.com/SebastianPinzonC/Operativos-2023`
-
- Documentación oficial de Python 3: `https://docs.python.org/3/`
-
- Streamlit Documentation: `https://docs.streamlit.io/`
-
- Pygame Documentation: `https://www.pygame.org/docs/`
-
- Docker Documentation: `https://docs.docker.com/`