Prácticas

Práctica 1: Debate sobre Aplicaciones

- **Tema**: "¿Es la arquitectura Harvard aún relevante en la era de los sistemas multicore?"
- **Recursos: Investigar casos como FPGAs, IoT y procesadores RISC-V.**

Práctica 2: Análisis de Cuellos de Botella

- **Herramienta**: Software de profiling (Ej: Valgrind).
- https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html#gs.kt03in
- https://docs.qualcomm.com/bundle/publicresource/topics/80-71528-1/snapdragon_profiler.html?product=1601111740010426

Porque es importante realizar profiling a una aplicación?

Lo estas haciendo actualmente?

Práctica 3: Análisis de Cuellos de Botella con Python

Análisis de Rendimiento en Python con cProfile y SnakeViz

- Identificar funciones críticas que consumen más tiempo de ejecución.
- Aprender a optimizar código basado en datos cuantitativos.
- Visualizar resultados de manera gráfica e interactiva.

Herramientas Requeridas

1. Python 3.x: Descargar desde (https://www.python.org/downloads/).
2. cProfile: Módulo integrado en Python.
3. SnakeViz: Visualizador de perfiles. Instalar con bash:

pip install snakeviz

Ojo con la variable de entorno o path :

WARNING: The script snakeviz.exe is installed in 'C:\Users\cemol\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-packages\Python313\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.

4. Editor de Código: VS Code, PyCharm Community, o cualquier editor.

Paso 1: Crear un Programa de Ejemplo

Crea un script con dos funciones: una ineficiente y otra optimizada.

python

```
# Archivo: practica_profiler.py
import time

def funcion_lenta():
    lista = []
    for i in range(5000):
        lista.append(i)
        # Bucle redundante (ineficiencia)
        for j in range(len(lista)):
            lista[j] += 0.5
    return lista

def funcion_optimizada():
    return [i + 0.5 for i in range(5000)]

if __name__ == "__main__":
    inicio = time.time()
    funcion_lenta()
    funcion_optimizada()
    print(f"Tiempo total: {time.time() - inicio:.2f} segundos")
```

Paso 2: Ejecutar el Profiler (cProfile)

1. Abre una terminal (Command Prompt o PowerShell) y navega a la carpeta del script.

2. Genera un archivo de perfil usando bash:

   python -m cProfile -o perfil.prof practica_profiler.py

   Esto creará un archivo `perfil.prof` con los datos de ejecución.

Paso 3: Visualizar Resultados con SnakeViz

1. Ejecuta SnakeViz para abrir una visualización interactiva usando bash:

   snakeviz perfil.prof

   ó
   C:\Users\cemol\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-packages\Python313\Scripts\snakeviz.exe .\perfil.prof

   Se abrirá automáticamente una pestaña en tu navegador con un diagrama de llamadas.

Paso 4: Interpretar los Resultados

En SnakeViz:
- Vista **Icicle**: Muestra la jerarquía de llamadas.
- Vista **Sunburst**: Visualiza el tiempo por función en forma de anillos.

Datos clave a observar:

1. Tiempo total en `funcion_lenta` vs. `funcion_optimizada`.
2. Número de llamadas a funciones como `list.append` o `range`.
3. Tiempo acumulado (incluyendo funciones internas de Python).

Paso 5: Análisis Crítico

Responde las siguientes preguntas:
- 1. ¿Qué función consume más tiempo? ¿Por qué?

- 2. ¿Cuántas veces se ejecuta el bucle interno de `funcion_lenta`?
- 3. ¿Cómo afecta la complejidad algorítmica (O(n²)) al rendimiento?

Paso 6: Optimizar el Código
Modifica `funcion_lenta` para eliminar el bucle redundante:

python

```
def funcion_lenta_optimizada():
    lista = []
    for i in range(5000):
        lista.append(i + 0.5)  # Evita el bucle interno
    return lista
```

Paso 7: Comparar Resultados

1. Vuelve a ejecutar el profiler con el código optimizado.
2. Compara los tiempos en SnakeViz.

Resultados esperados:

Antes de optimizar:

- `funcion_lenta`: ~95% del tiempo.
- - Miles de llamadas a `list.append` y `range`.

-Después de optimizar:

- funcion_lenta_optimizada`: Tiempo similar a `funcion_optimizada`.

lternativa: Usar PyCharm (Community Edition)

Si prefieres un IDE:

1. Abre el script en PyCharm.
2. Haz clic derecho en el archivo y selecciona *Run 'practica_profiler' with Profiler´*
3. Visualiza los resultados en la pestaña Profiler.

- cProfile es una herramienta poderosa para identificar cuellos de botella en Python.
- SnakeViz facilita la interpretación visual de datos complejos.
- La optimización basada en evidencia mejora drásticamente el rendimiento.

Recursos Adicionales:

- Documentación de cProfile (https://docs.python.org/3/library/profile.html).
- Tutorial de SnakeViz (https://jiffyclub.github.io/snakeviz/).
- Guía de PyCharm Profiler (https://www.jetbrains.com/help/pycharm/profiler.html).