Universidad EAFIT

Departamento de Informática y Sistemas

# ST0257 – Sistemas Operativos Proyecto 3: Paralelismo en un ambiente controlado

Diego Alexander Munera
Alejandro Torres Muñoz

Profesor: MBA, I.S. José Luis Montoya Pareja

Medellín, Colombia, Suramérica
November 4, 2024

# Contents

# Resumen

El paralelismo permite que las tareas se puedan realizar de manera más eficiente si se programa adecuadamente. Usando elementos y conceptos vistos en el curso, se implementaron patrones de programación concurrente y paralela que optimizan el procesamiento de archivos CSV en diferentes modos de ejecución.

# Palabras Clave

Paralelismo, procesamiento en paralelo, concurrencia, eficiencia, sistemas operativos.

# Chapter 1

# Introducción

Este proyecto explora el procesamiento de archivos de gran tamaño mediante paralelismo y concurrencia en Python, con el objetivo de mejorar la eficiencia y el uso de recursos del sistema. La implementación fue realizada en tres modos de ejecución que comparan diferentes enfoques de paralelismo:

- Procesamiento secuencial.
- Paralelismo limitado a un solo núcleo (single core).
- Paralelismo distribuido en múltiples núcleos (multi-core).

Para cada archivo CSV, se implementó una lectura en fragmentos, lo cual permite reducir el uso de memoria y manejar archivos grandes sin problemas. Además, se incluyó un proceso de detección de codificación de los archivos, ya que los datos pueden provenir de distintas fuentes y tener diferentes formatos. Sin embargo, esta detección introduce un pequeño retraso adicional que afecta el tiempo total de procesamiento.

# Chapter 2

# Descripción del Proyecto

## 2.1 Descripción de los Códigos

### 2.1.1 `load_sequential.py`

Este código lee los archivos secuencialmente, cargando cada archivo en memoria antes de pasar al siguiente. La función `load_files_sequential` abre y procesa cada archivo individualmente, y los datos se cargan en fragmentos para optimizar el uso de memoria. Durante cada paso, se monitorea el uso de recursos.

### 2.1.2 `load_single_core.py`

Este código usa un `ThreadPoolExecutor` para procesar múltiples archivos simultáneamente, pero limita el procesamiento a un solo núcleo. Cada archivo se asigna a un hilo que lee los datos en fragmentos. La función `process_file` maneja el procesamiento de cada archivo y monitorea el uso de memoria.

### 2.1.3 `load_multi_core.py`

En este modo, se utiliza un `ProcessPoolExecutor` para procesar archivos en paralelo, distribuyendo la carga en múltiples núcleos. Cada proceso adicionalmente crea hilos para leer el archivo en fragmentos, lo cual maximiza la eficiencia del procesamiento.

# Chapter 3

# Resultados y Análisis

A continuación se presentan los resultados obtenidos en cada uno de los tres modos de ejecución.

Table 3.1: Resultados del procesamiento de archivos - Modo Secuencial

| **Nombre** | **T. Inicial** | **T. Final** | **Duración (ms)** | **Memoria (MB)** |
|---|---|---|---|---|
| files/MXvideos.csv | 15:26:05.908 | 15:26:06.494 | 586.056 | 58.16 |
| files/INvideos.csv | 15:26:06.495 | 15:26:07.201 | 706.672 | 83.8 |
| files/DEvideos.csv | 15:26:07.201 | 15:26:08.018 | 816.561 | 107.5 |
| files/JPvideos.csv | 15:26:08.018 | 15:26:08.479 | 461.272 | 117.17 |
| files/KRvideos.csv | 15:26:08.479 | 15:26:08.982 | 502.361 | 134.86 |
| files/CAvideos.csv | 15:26:08.982 | 15:26:09.717 | 735.109 | 153.31 |
| files/RUvideos.csv | 15:26:09.717 | 15:26:10.229 | 511.957 | 173.7 |
| files/FRvideos.csv | 15:26:10.229 | 15:26:10.869 | 639.371 | 192.45 |
| files/USvideos.csv | 15:26:10.869 | 15:26:11.590 | 721.553 | 210.2 |
| files/GBvideos.csv | 15:26:11.590 | 15:26:12.211 | 620.831 | 227.73 |

Tiempo total del programa: 6.31 segundos

Table 3.2: Resultados del procesamiento de archivos - Modo Single Core

| Nombre | T. Inicial | T. Final | Duración (ms) | Memoria (MB) |
|---|---|---|---|---|
| files/FRvideos.csv | 15:26:25.357 | 15:26:29.011 | 3654.46 | 147.61 |
| files/GBvideos.csv | 15:26:25.386 | 15:26:30.422 | 5035.26 | 201.8 |
| files/USvideos.csv | 15:26:25.357 | 15:26:30.484 | 5127.03 | 203.38 |
| files/CAvideos.csv | 15:26:25.356 | 15:26:30.518 | 5161.05 | 204.78 |
| files/INvideos.csv | 15:26:25.356 | 15:26:30.672 | 5316.35 | 209.06 |
| files/MXvideos.csv | 15:26:25.356 | 15:26:30.820 | 5464.59 | 215.08 |
| files/DEvideos.csv | 15:26:25.356 | 15:26:30.878 | 5522.36 | 217.33 |
| files/KRvideos.csv | 15:26:25.356 | 15:26:31.100 | 5743.71 | 229.44 |
| files/RUvideos.csv | 15:26:25.357 | 15:26:31.438 | 6081.9 | 245.2 |
| files/JPvideos.csv | 15:26:25.356 | 15:26:31.449 | 6092.81 | 245.78 |

Tiempo total del programa: 6.10 segundos

Table 3.3: Resultados del procesamiento de archivos - Modo Multi Core

| Nombre | T. Inicial | T. Final | Duración (ms) | Memoria (MB) |
|---|---|---|---|---|
| files/MXvideos.csv | 15:26:44.289 | 15:26:45.138 | 848.769 | 141.7 |
| files/INvideos.csv | 15:26:44.291 | 15:26:45.371 | 1079.47 | 162 |
| files/DEvideos.csv | 15:26:44.294 | 15:26:45.317 | 1023.03 | 174.27 |
| files/JPvideos.csv | 15:26:44.296 | 15:26:44.913 | 616.516 | 92.52 |
| files/KRvideos.csv | 15:26:44.300 | 15:26:45.035 | 735.233 | 124.16 |
| files/CAvideos.csv | 15:26:44.300 | 15:26:45.352 | 1051.59 | 171.66 |
| files/RUvideos.csv | 15:26:44.301 | 15:26:45.155 | 853.54 | 156.81 |
| files/FRvideos.csv | 15:26:44.304 | 15:26:45.283 | 979.281 | 160.36 |
| files/USvideos.csv | 15:26:44.305 | 15:26:45.339 | 1033.34 | 169.83 |
| files/GBvideos.csv | 15:26:44.309 | 15:26:45.226 | 917.679 | 148.48 |

Tiempo total del programa: 1.29 segundos

## 3.1 Análisis de Resultados

En el modo secuencial, el tiempo total de procesamiento es mayor debido a que los archivos se leen uno tras otro sin aprovechar el paralelismo. En el modo single core, se mejora el tiempo al procesar múltiples archivos en paralelo, aunque el límite de un solo núcleo restringe la ganancia en velocidad. Finalmente, el modo multi-core muestra la mayor eficiencia, con tiempos de

ejecución significativamente menores, ya que se utilizan todos los núcleos disponibles en el equipo.

# Chapter 4

# Conclusiones

- La implementación de paralelismo y concurrencia permite reducir el tiempo de procesamiento de archivos grandes.
- La detección de codificación, aunque necesaria para asegurar la correcta lectura de archivos, introduce un retraso adicional en el tiempo total de procesamiento.
- El uso de múltiples núcleos en el modo multi-core mejora considerablemente el rendimiento en comparación con los otros modos.