UNIVERSIDAD TECNOLÓGICA DEL URUGUAY
ITR SUROESTE · FRAY BENTOS
INGENIERÍA MECATRÓNICA
UC de Programación III · 2025
Profesores: Giovani Bolzan Cogo y Mariano Arbiza

| Práctico de laboratorio IV - Concurrencia vía hilos | Periodo: de 6/11/2025 hasta 16/11/2025 |
|---|---|


**Integrantes**

Guido Andreoli – 5.307.933-9
Lucas Lopez – 5.259.876-6
Hector Pereira – 5.582.582-5
Alvaro Kuster – 3.832.334-1



**Resumen**

(Resumen breve del trabajo práctico: objetivos generales, descripción del sistema o problema a resolver y resultados esperados.)


# 1. Introducción

(Explicación general del tema, contexto, motivación y relación con la unidad curricular.)

## 1.1. Objetivos

- (Objetivo específico 1)
- (Objetivo específico 2)
- (Objetivo específico 3)

# 2. Fundamentación Teórica

(Explicación de conceptos teóricos relacionados: bibliotecas, módulos, estructuras, algoritmos, etc.)

## 2.1. Concepto 1

(Desarrollo.)

## 2.2. Concepto 2

(Desarrollo.)

# 3. Metodología

## 3.1. Herramientas y material

(Detallar entorno de desarrollo, lenguajes, librerías, equipos y materiales utilizados.)

## 3.2. Procedimiento

(Describir paso a paso el desarrollo del trabajo práctico, estructura del código y funcionamiento general del sistema.)

## 4. Resultados

(Exposición de los resultados obtenidos, comparación de versiones, mediciones, análisis, etc.)

Tabla 1: (Ejemplo de tabla de resultados o comparación de tiempos)

| Prueba | Caso 1 | Caso 2 | Diferencia (%) |
|---|---|---|---|
| (Descripción) | – | – | – |

## 5. Conclusiones

(Conclusiones principales, observaciones, limitaciones y posibles mejoras.)

## A. Recursos

- **Carpeta general (Drive):** (enlace)
- **Código fuente:** (enlaceaColaboGitHub)
- **Overleaf:** (enlace)

## B. Código fuente

```
# Librerias

import os
import time
import multiprocessing
import threading
import sympy as sp
from typing import List, Tuple


# Conteo de palabras


## Paralelo


def count_words_worker(filename: str, q: multiprocessing.Queue) -> None:
    """Funcion que se ejecuta en cada proceso."""
    t0 = time.time()
    try:
        with open(filename, "r", encoding="utf-8") as f:
            count = len(f.read().split())
        q.put((filename, count, time.time() - t0))
    except Exception as e:
        q.put((filename, f"ERROR: {e}", 0.0))

def count_words_in_files_parallel(files: List[str]) -> Tuple[List[Tuple[str, int, float]],
    float]:
    """Cuenta palabras en varios archivos de forma paralela usando multiprocessing."""
    q = multiprocessing.Queue()
    processes = [multiprocessing.Process(target=count_words_worker, args=(f, q)) for f in
    files]
```

```
    start_total = time.time()
    [p.start() for p in processes]
    [p.join() for p in processes]
    total_time = time.time() - start_total

    results = [q.get() for _ in processes]
    return results, total_time


# ------------------------------------------------------------------------
# Testeo
# ------------------------------------------------------------------------

if __name__ == "__main__":
    input_dir = "archivos_texto"

    if not os.path.isdir(input_dir):
        print(f"La carpeta '{input_dir}' no existe.")
    else:
        files = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith(".
    txt")]

        print(f"Archivos detectados: {files}")
        print("Ejecutando conteo en paralelo...\n")

        results, total_time = count_words_in_files_parallel(files)

        print("=== Resultados individuales ===")
        for filename, count, elapsed in results:
            if isinstance(count, int):
                print(f"  {os.path.basename(filename)} -> {count} palabras ({elapsed:.4f} s)
    ")
            else:
                print(f"  {os.path.basename(filename)} -> {count}")

        print(f"\nTiempo total de ejecucion (paralelo): {total_time:.4f} s")


## Secuencial


def count_words_in_files(files: List[str]) -> Tuple[List[Tuple[str, int, float]], float]:
    """
    Cuenta las palabras de varios archivos de texto de forma secuencial.
    Retorna una lista con (nombre, cantidad_palabras, tiempo_individual) y el tiempo total.
    """
    results = []
    start_total = time.time()

    for filename in files:
        t0 = time.time()
        try:
            with open(filename, "r", encoding="utf-8") as f:
                word_count = len(f.read().split())
            elapsed = time.time() - t0
            results.append((filename, word_count, elapsed))
        except Exception as e:
            results.append((filename, f"ERROR: {e}", 0.0))

    total_time = time.time() - start_total
    return results, total_time
```

```
# ------------------------------------------------------------------------
# Testeo
# ------------------------------------------------------------------------

if __name__ == "__main__":
    # Ruta de prueba (cambia esto por tu carpeta con archivos)
    input_dir = "archivos_texto"

    # Filtramos solo archivos .txt
    if not os.path.isdir(input_dir):
        print(f"La carpeta '{input_dir}' no existe.")
    else:
        files = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith(".
    txt")]

        print(f"Archivos detectados: {files}")
        print("Ejecutando conteo secuencial...\n")

        results, total_time = count_words_in_files(files)

        # Imprimir resultados detallados
        print("=== Resultados individuales ===")
        for filename, count, elapsed in results:
            if isinstance(count, int):
                print(f"  {os.path.basename(filename)} -> {count} palabras ({elapsed:.4f} s)
    ")
            else:
                print(f"  {os.path.basename(filename)} -> {count}")

        print(f"\nTiempo total de ejecucion: {total_time:.4f} s")


# Calculo de derivadas e integrales


## Secuencial
def solve_equation_sequential(equation: str):
    """Resuelve una sola ecuacion de forma secuencial y devuelve el resultado y tiempo."""
    start = time.time()
    try:
        x, a = sp.symbols('x a')
        expr = sp.sympify(equation)
        integral = sp.integrate(expr, x)
        elapsed = time.time() - start
        return str(expr), str(integral), elapsed
    except Exception as e:
        elapsed = time.time() - start
        return equation, f"ERROR: {e}", elapsed


# ------------------------------------------------------------------------
# Testeo
# ------------------------------------------------------------------------

# --- Cargar ecuaciones desde archivo ---
try:
    with open("ecuaciones.txt", "r", encoding="utf-8") as f:
        ecuaciones_a_resolver = [line.strip() for line in f if line.strip()]
except FileNotFoundError:
```

```
    print("Error: No se encontro el archivo 'ecuaciones.txt'.")
    raise SystemExit(1)

if not ecuaciones_a_resolver:
    print("El archivo 'ecuaciones.txt' esta vacio.")
    raise SystemExit(0)

# --- Resolver ecuaciones secuencialmente ---
t0_total = time.time()
resultados = []
for eq_str in ecuaciones_a_resolver:
    resultados.append(solve_equation_sequential(eq_str))
t_total = time.time() - t0_total

# --- Imprimir resultados ---
print("\n--- Resultados de las Integrales (Secuencial) ---")
for i, (ecuacion, resultado, elapsed) in enumerate(resultados, start=1):
    print(f"  [Ecuacion {i}] {ecuacion}")
    print(f"               Integral: {resultado}")
    print(f"               Tiempo: {elapsed:.6f} s\n")

print(f"Tiempo total (lote): {t_total:.6f} s")


## Paralelo


# --- Pipe (lectura, escritura) y Lock para proteger .send() ---
read_pipe, write_pipe = multiprocessing.Pipe(duplex=False)
pipe_lock = threading.Lock()


def solve_equation(equation: str, pipe_to_write, lock: threading.Lock) -> None:
    hilo_actual = threading.current_thread().name
    start = time.time()  # medir tiempo por ecuacion (sympify + integrate)
    try:
        x, a = sp.symbols('x a')
        expr = sp.sympify(equation)
        integral = sp.integrate(expr, x)
        elapsed = time.time() - start

        # Enviamos el resultado de forma protegida
        with lock:
            pipe_to_write.send((str(expr), str(integral), elapsed, hilo_actual))
    except Exception as e:
        elapsed = time.time() - start
        # Log al stdout
        print(f"[{hilo_actual}] Error resolviendo '{equation}': {e}")
        # Intentamos reportar el error por el pipe; si ya esta cerrado, lo ignoramos
        try:
            with lock:
                pipe_to_write.send((equation, f"ERROR: {e}", elapsed, hilo_actual))
        except OSError:
            pass

def solve_equations_parallel(equations):
    """Lanza un hilo por ecuacion y devuelve todos los resultados + tiempo total."""
    read_pipe, write_pipe = multiprocessing.Pipe(duplex=False)
    pipe_lock = threading.Lock()
    hilos = []
    t0_total = time.time()
```

```
    # Lanzar hilos
    for i, eq_str in enumerate(equations, start=1):
        hilo = threading.Thread(
            target=solve_equation,
            args=(eq_str, write_pipe, pipe_lock),
            name=f"Trabajador-{i}",
            daemon=True,
        )
        hilos.append(hilo)
        hilo.start()

    for hilo in hilos:
        hilo.join()

    write_pipe.close()

    results = []
    for _ in range(len(equations)):
        ecuacion_original, resultado, elapsed, hilo_nombre = read_pipe.recv()
        results.append((ecuacion_original, resultado, elapsed, hilo_nombre))

    read_pipe.close()
    total_time = time.time() - t0_total
    return results, total_time

# ------------------------------------------------------------------------
# Testeo
# ------------------------------------------------------------------------

if __name__ == "__main__":

    # --- Cargar ecuaciones desde archivo ---
    try:
        with open("ecuaciones.txt", "r", encoding="utf-8") as f:
            ecuaciones_a_resolver = [line.strip() for line in f if line.strip()]
    except FileNotFoundError:
        print("Error: No se encontro el archivo 'ecuaciones.txt'.")
        raise SystemExit(1)

    if not ecuaciones_a_resolver:
        print("El archivo 'ecuaciones.txt' esta vacio.")
        raise SystemExit(0)

    # --- Ejecutar funcion paralela ---
    resultados, t_total = solve_equations_parallel(ecuaciones_a_resolver)

    # --- Mostrar resultados ---
    print("\n--- Resultados de las Integrales (Paralelo) ---")
    for expr, integral, elapsed, hilo in resultados:
        print(f"  [{hilo}] Ecuacion: {expr}")
        print(f"               Integral: {integral}")
        print(f"               Tiempo: {elapsed:.6f} s\n")

    print(f"Tiempo total (lote): {t_total:.6f} s")



# Integracion final
```

```
# ------------------------------------------------------------------------
# Funciones auxiliares de presentacion
# ------------------------------------------------------------------------

def print_word_results(title: str, results, total_time: float):
    print(f"\n=== {title} ===")
    for filename, count, elapsed in results:
        print(f"  {os.path.basename(filename)} -> {count} palabras ({elapsed:.4f} s)")
    print(f"Tiempo total: {total_time:.4f} s")

def print_equation_results(title: str, results, total_time: float, paralelo=False):
    print(f"\n=== {title} ===")
    if paralelo:
        for expr, integral, elapsed, hilo in results:
            print(f"  [{hilo}] {expr} -> {integral} ({elapsed:.4f} s)")
    else:
        for expr, integral, elapsed in results:
            print(f"  {expr} -> {integral} ({elapsed:.4f} s)")
    print(f"Tiempo total: {total_time:.4f} s")

def speedup(ts: float, tp: float) -> str:
    return f"x{ts/tp:.2f}" if tp > 0 else "N/A"

# ------------------------------------------------------------------------
# Programa comparador
# ------------------------------------------------------------------------

if __name__ == "__main__":
    # -------------------------------------
    # 1. Conteo de palabras
    # -------------------------------------
    input_dir = "archivos_texto"
    files = [os.path.join(input_dir, f) for f in os.listdir(input_dir) if f.endswith(".txt")
    ] if os.path.isdir(input_dir) else []

    if files:
        # Secuencial
        seq_word_results, seq_word_time = count_words_in_files(files)

        # Paralelo
        par_word_results, par_word_time = count_words_in_files_parallel(files)

        # Resultados
        print_word_results("Conteo de palabras (Secuencial)", seq_word_results,
    seq_word_time)
        print_word_results("Conteo de palabras (Paralelo)", par_word_results, par_word_time)
        print(f"\nSpeedup conteo de palabras: {speedup(seq_word_time, par_word_time)}")
    else:
        print("No se encontraron archivos .txt para el conteo de palabras.")

    # -------------------------------------
    # 2. Resolucion de ecuaciones
    # -------------------------------------
    equations = []
    if os.path.exists("ecuaciones.txt"):
        with open("ecuaciones.txt", "r", encoding="utf-8") as f:
            equations = [line.strip() for line in f if line.strip()]

    if equations:
        # Secuencial
        seq_eq_results = []
```

```
        seq_start = time.time()
        for eq in equations:
            seq_eq_results.append(solve_equation_sequential(eq))
        seq_eq_time = time.time() - seq_start

        # Paralelo
        par_eq_results, par_eq_time = solve_equations_parallel(equations)

        # Resultados
        print_equation_results("Resolucion de ecuaciones (Secuencial)", seq_eq_results,
    seq_eq_time)
        print_equation_results("Resolucion de ecuaciones (Paralelo)", par_eq_results,
    par_eq_time, paralelo=True)
        print(f"\nSpeedup resolucion de ecuaciones: {speedup(seq_eq_time, par_eq_time)}")
    else:
        print("No se encontraron ecuaciones en ecuaciones.txt.")
```

## C. Salidas de consola

(Aquí se insertan los resultados de consola, salidas de prueba, logs o verificaciones.)