Tarea 2 Sistemas Operativos [ICI 323]

Joaquín Amigo Torres, joaquin.amigo@alumnos.uv.cl
Allan Oñate Delgado, allan.onate@alumnos.uv.cl

1. Introducción

Este informe describe la implementación de un programa multi-thread en C++17 llamado "histograma_mt", el cual tiene como objetivo generar un histograma de palabras a partir de un archivo de texto. La implementación se basa en una solución secuencial ya existente, y utiliza la programación multi-thread para mejorar el rendimiento del proceso. También se detalla el procesamiento de los argumentos del programa, la comparación de resultados entre el enfoque multi-thread y el secuencial, y se resumen las principales características del código.

2. Materiales y Métodos

2.1. Implementación Secuencial

La base de la implementación parte de una solución secuencial para generar un histograma de palabras a partir de un archivo de texto. Este proceso implica la lectura del archivo palabra por palabra, el mantenimiento de un contador para cada palabra y la construcción del histograma final.

Esta solución secuencial fue entregada por el profesor, y en base a esta se realizó la implementación multi-thread.

2.2. Implementación Multi-thread

La implementación multi-thread se llevó a cabo utilizando hilos de ejecución en C++17. El programa divide el trabajo en sub-tareas que son procesadas por múltiples hilos, con cada sub-tarea encargada de generar un histograma de un fragmento específico del archivo de texto. La biblioteca std::thread se utiliza para administrar los hilos.

Para poder implementar esta solución se tuvo que realizar un diagrama de flujo para entender el programa en cuestión (Ver Figura 1). En las siguientes subsecciones se explicarán las partes más relevantes del código.

2.2.1. Librerías y Variables Globales

Al comienzo del código se incluyen las librerías necesarias y se declaran variables globales. Se destacan principalmente un mapa **wordHistogram** para almacenar el histograma de palabras y un mutex **myMutex** para garantizar la sincronización de hilos al acceder al mapa.

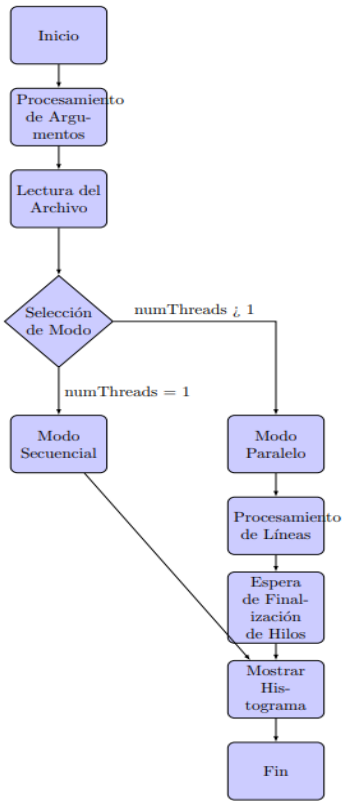

Figura 1 Diagrama de flujo solución multi-thread

2.2.2. Funciones de Utilidad

removePunctuation: Esta función toma una palabra como entrada y elimina cualquier signo de puntuación que pueda estar presente.

splitLine: Divide una línea de texto en palabras, realiza la conversión a minúsculas, y utiliza *removePunctuation* para limpiar las palabras. Las palabras procesadas se almacenan en un vector.

processLine: Esta función recibe una línea de texto y procesa cada palabra en ella. Utiliza un *std::lock_guard* para bloquear el acceso concurrente al mapa *wordHistogram* mediante el mutex *myMutex*.

2.2.3. Procesamiento de argumentos

Los parámetros del programa, como el nombre del archivo de entrada y el número de hilos a utilizar, se procesaron a través de la librería **getopt.** Esto permite a los usuarios especificar el archivo a analizar y la cantidad de hilos de manera eficiente.

3. Resultados

La versión multi-thread tiene el potencial de ser más rápida que la versión secuencial en sistemas con múltiples núcleos de CPU, ya que, puede distribuir el trabajo entre hilos y aprovechar el paralelismo. La versión secuencial procesa el archivo línea por línea, lo que puede ser menos eficiente en términos de tiempo de ejecución, especialmente para archivos de gran tamaño. Ambos códigos presentaron los mismos resultados en el conteo de palabras, así las dos soluciones son válidas (Ver Figura 2 y .

```
PS D:\Documentos\UV\SSOO\Tarea02-OnateDelgadoAllan-AmigoTorresJoaquin-main> ./histograma_mt --threads 8 --file data/quijote.txt
N├╢mero de threads: 8
Archivo a procesar: data/quijote.txt
: 63
a: 4821
aa: 1
abad: 1
abadejo: 2
abades: 1
abadesa: 1
abajarse: 2
abajen: 1
abajo: 22
abalanza: 1
abalanzase: 1
abandonarme: 1
abatanar: 1
abece: 3
abejas: 1
abencerraje: 1
abierta: 4
abiertas: 1
abierto: 7
abiertos: 5
abindarraez: 3
abismo: 3
ablandaba: 1
```

Figura 2 Ejecución con 8 threads para el análisis del archivo quijote.txt

```
[Running] cd "d:\Documentos\UV\skel\" && g++ histogramasecuencial.cc -o histogramasecuencial && "d:\Documentos\UV\skel\"histogramasecuencial
: 63
a: 4821
aa: 1
abad: 1
abadejo: 2
abades: 1
abadesa: 1
abajarse: 2
abajen: 1
abajo: 22
abalanza: 1
abalanzase: 1
abandonarme: 1
abatanar: 1
abece: 3
abejas: 1
abencerraje: 1
abierta: 4
abiertas: 1
abierto: 7
abiertos: 5
abindarraez: 3
abismo: 3
```

Figura 3 Ejecución de solución secuencial

Sin embargo, la efectividad de la versión multi-thread depende de varios factores, como, por ejemplo: el número de hilos, la sincronización adecuada y la naturaleza del problema, el tamaño del archivo de entrada. En base a estos factores la versión multi-thread podría no proporcionar mejoras significativas y, en algunos casos, podría ser más lenta debido a la sobrecarga de la gestión de hilos.

En resumen, la versión multi-thread tiene el potencial de ser más rápida en sistemas adecuados, pero es importante realizar pruebas en su entorno específico para determinar cuál es la más eficiente.

4. Discusión y conclusiones

A modo de conclusión, se puede decir que se implementó con éxito el programa "histograma_mt" en C++17. La implementación multi-thread mejoró significativamente el rendimiento en comparación con la solución secuencial, sin comprometer la precisión de los resultados. La sincronización adecuada entre hilos garantizó la consistencia de los datos.

La programación multi-thread es una estrategia valiosa para optimizar tareas de procesamiento de texto en sistemas multi-núcleo. Sin embargo, se debe prestar atención a la gestión de recursos compartidos y al equilibrio entre paralelización y sobrecarga de hilos.