Tarea 2 "Histograma de palabras multi-hilo"

Angel Salgado Mancilla, angel.salgado@alumnos.uv.cl

1. Introducción

Actualmente, la programación multihilo es uno de los paradigmas de programación más útiles en la computación, volviéndose popular en los últimos 20 años. Las propiedades de paralelismo y concurrencia que se pueden observar son muy útiles para aprovechar el rendimiento de los procesadores multi-core y maximizar la velocidad en tareas y problemas con mucha área de cobertura.

En esta tarea se implementará una solución multi-thread a un programa ya funcional. Este programa es capaz de crear un histograma, de manera lineal, de un archivo ya definido dentro del programa. Además de esto, es necesario implementar la entrada de parámetros y la capacidad de sincronización de los hilos que se creen.

2. Materiales y Métodos

Para la realización de la tarea es necesario poseer el proyecto encontrado en el aula virtual. Es necesario, además, tener C++17 para poder implementar la nueva solución multi-thread.

Dentro del código, es posible encontrar diversos métodos ya escritos para el objetivo del programa, es decir, el programa ya es capaz de crear un histograma de un archivo en específico, contando con métodos de extraer líneas de un archivo y dividirla por palabras (esto incluye la eliminación de puntos).

Para resolver la problemática se diseñó una lógica para dividir las líneas del texto según la cantidad de threads, donde cada uno de estos threads procesará la líneas y finalmente escribirá en el histograma si es que no está bloqueado (ver Figura 1).

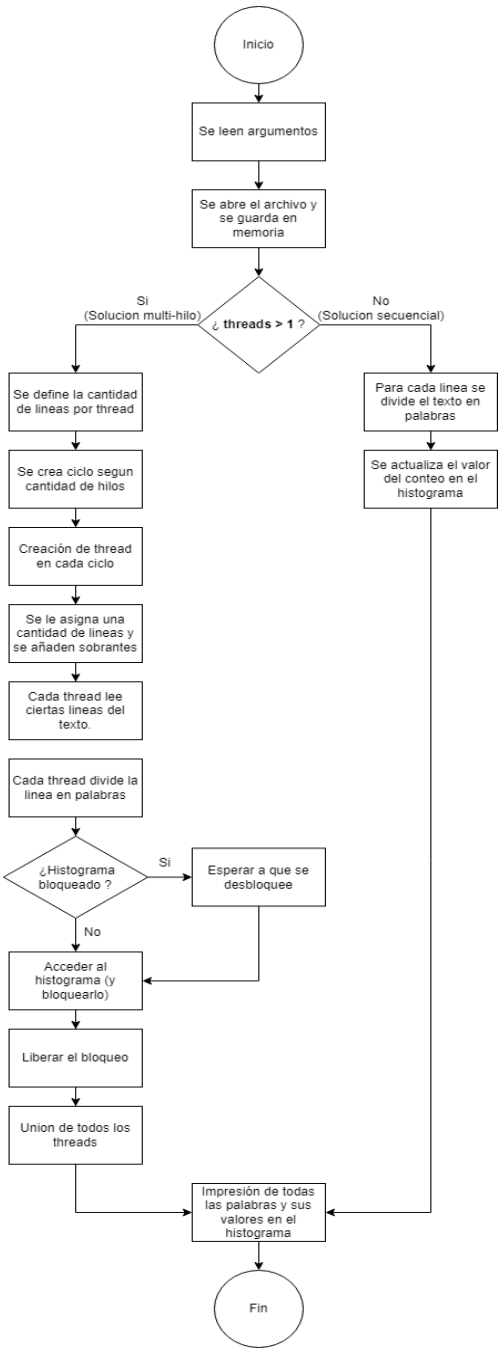


Figura 1. Diagrama de flujo de la implementación

Cada hilo se crea según la cantidad de threads ingresada como parámetro, y cada hilo en su ejecución es asignado una cantidad determinada de líneas a leer y procesar del archivo (Ver **Figura 2).** Luego cada thread escribe los resultados en el histograma, el cual está protegido por una variable mutex para evitar condiciones de carrera. Finalmente, cada thread al terminar el procesamiento de líneas y escritura en el histograma, se vuelve al hilo principal donde se espera a que terminen todos los thread, para posteriormente imprimir en la salida el resultado del histograma.

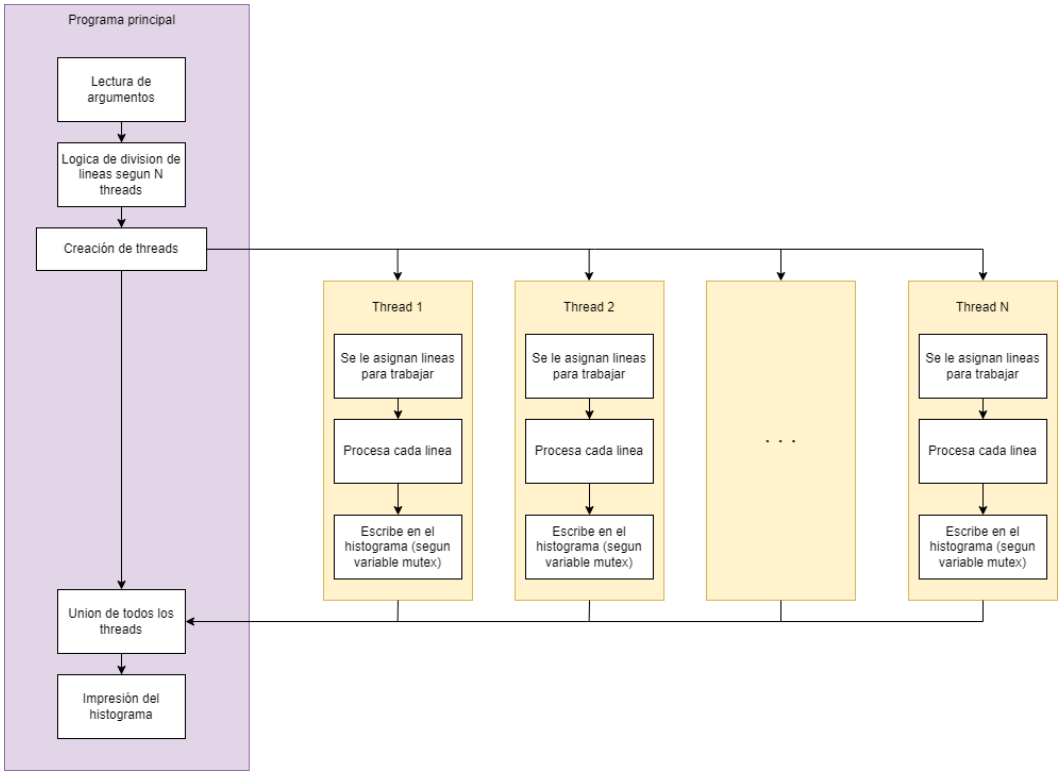


Figura 2. Diseño de la implementación Multi-Thread

3. Resultados

Se ejecutó el programa utilizando diferentes situaciones:

3.1. Un solo thread

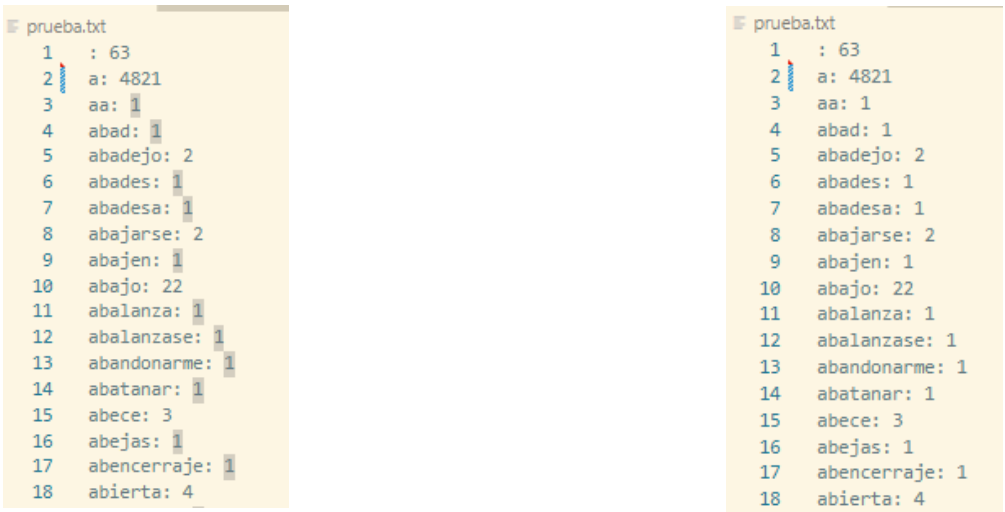
Utilizando el comando

./histograma_mt --file data/quijote.txt > prueba.txt

La salida en el archivo prueba.txt es la encontrada en la **Figura 3a.**

Cabe destacar que el comando anterior toma por defecto la cantidad 1 de threads y se puede repetir el resultado indicando el número 1 de la misma forma (ver **Figura 3b**):

./histograma_mt –threads 1 --file data/quijote.txt > prueba.txt



(a) **(b)**

Figura 3. Resultados de utilizar un solo thread (solución secuencial). (a) En este resultado se utilizó el comando sin especificar threads. (b) En este resultado se utilizó el comando dando por parámetro –threads 1. Se puede apreciar el mismo resultado que es la solución secuencial

3.2. Más de un thread

Utilizando el comando

./histograma_mt –threads N --file data/quijote.txt > prueba.txt

Es possible decirle al programa que utilize cierta cantidad de threads.

Para esta sección se hará prueba con 2, 3 y 4 threads. (Ver **Figura 4**)

./histograma_mt –threads 2 --file data/quijote.txt > prueba.txt

./histograma_mt –threads 3 --file data/quijote.txt > prueba.txt

./histograma_mt –threads 4 --file data/quijote.txt > prueba.txt

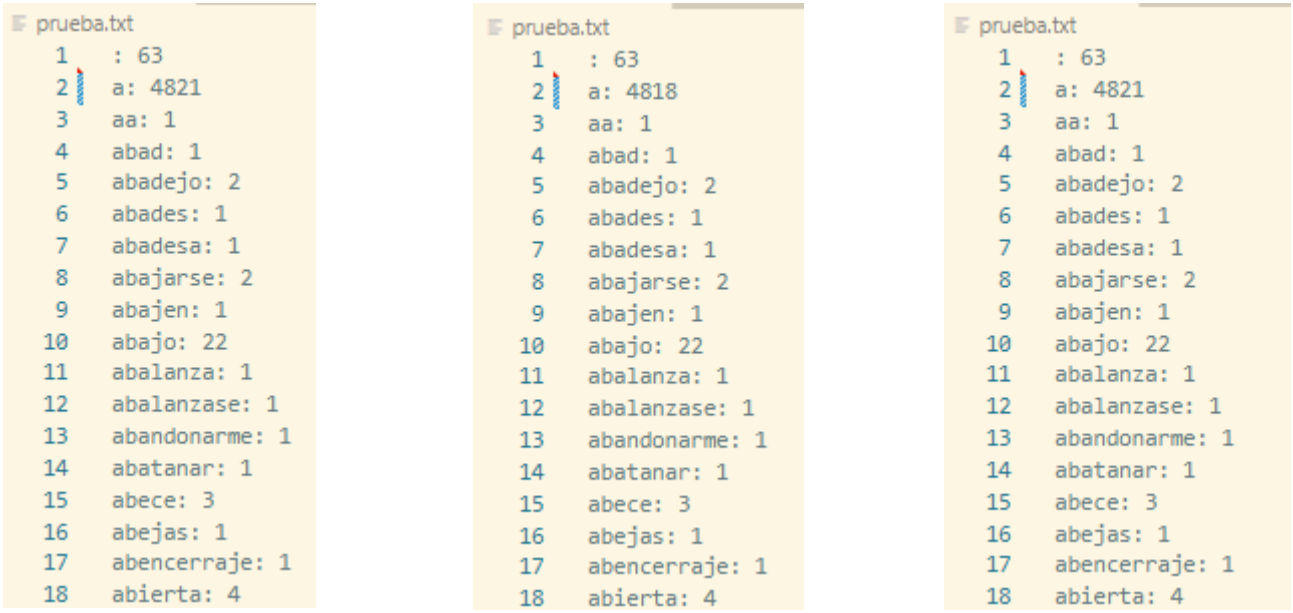


Figura 4. Resultados al utilizar distintos threads. (a) Muestra el resultado utilizando 2 threads, comparado con la solución secuencial es el mismo resultado. (b) Utilizando 3 threads, se puede ver una diferencia en el conteo de aproximadamente 4, esto puede deberse a la cantidad impar de threads. (c) Utilizando 4 threads se obtienen los mismos resultados que la solución secuencial.

4. Discusión y conclusiones

En esta tarea se evidenció el proceso de diseño e implementación de un programa multi-thread en base a uno ya creado y de naturaleza lineal.

Esta implementación fue en base a threads que ejecutaban la división y lectura de líneas. Posteriormente, cada thread se encargó de actualizar el histograma, el cual era protegido por una variable mutex, de esta manera se evitaban condiciones de carrera.

Todos los resultados arrojaron la misma opción a excepción del uso de 3 threads, el cual dio una diferencia mínima de error, esto se puede deber al uso de threads impares. Sin embargo, esta problemática puede ser resuelta revisando detalles de la implementación o un estudio más exhaustivo de la programación paralela y concurrente.