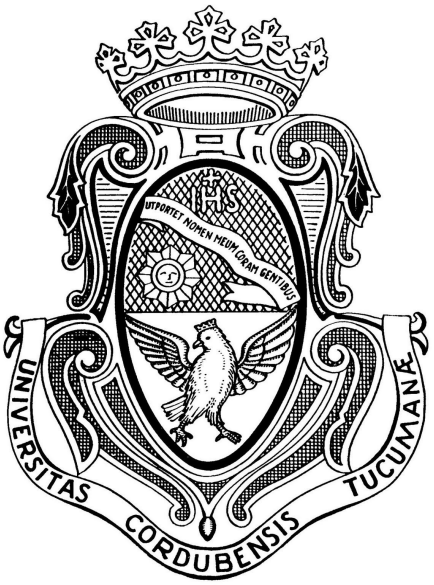
UNC Universidad Nacional de Córdoba

FCEFyN Facultad de Ciencias Exactas Físicas y Naturales

UNIVERSIDAD NACIONAL DE CÓRDOBA
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES



PROGRAMACIÓN CONCURRENTE

Trabajo práctico N°1

Sistema de gestión de imágenes

Comisión: Única
Docentes: VENTRE, Luis Orlando; LUDEMANN Mauricio
Fecha de entrega: 24/04/2023

Grupo: Los Power Rangers		
Apellido	**Nombre**	**Matrícula**
GUGLIELMOTTI	Bruno	43474558
OTALORA	Joaquín	43553345
PAVÓN	Valentina	43134075
RICHTER	Federico	43421890
RODRÍGUEZ	Franco Aníbal	42994188

ÍNDICE

OBJETIVOS

- ➔ Realizar un diagrama de clases representativo del programa, donde puedan observarse todos sus componentes.
- ➔ Realizar un diagrama de secuencias donde puedan observarse las distintas interacciones al inicio (con el contenedor vacío), en el proceso 2 (un hilo intentando acceder a una imagen para mejorarla) y en el proceso 3 (un hilo intentando acceder a una imagen para recortarla).
- ➔ Resolver adecuadamente las situaciones de concurrencia, sorteando con éxito las dificultades en las secciones críticas.
- ➔ Lograr que el programa cumpla su cometido (desde el inicio hasta el final) en un tiempo mínimo de 10 segundos y máximo de 20 segundos.

INTRODUCCIÓN

En el contexto de un sistema de gestión de imágenes, se ha detectado que la funcionalidad encargada de mejorar la calidad de los archivos ha quedado obsoleta, por lo tanto, se ha solicitado el desarrollo de una nueva solución que tenga en cuenta ciertos lineamientos de diseño.

La nueva funcionalidad se divide en cuatro procesos que deben ser ejecutados de manera concurrente y eficiente. Cada proceso cuenta con un número determinado de hilos encargados de realizar tareas específicas.

El objetivo principal es mejorar la calidad de las imágenes, ajustarlas al tamaño final solicitado y almacenarlas en un contenedor final. Además, se debe llevar un registro de la cantidad de imágenes insertadas, mejoradas, ajustadas y finalizadas, así como también el estado de cada hilo del sistema.

En el presente informe se presentará la solución implementada y se describirán las principales estrategias utilizadas para garantizar la concurrencia y eficiencia de los procesos.

DESARROLLO

En primer lugar, el grupo analizó el planteo, debatió sobre cómo organizar las tareas y se escucharon las distintas opiniones y opciones propuestas por los integrantes.Una vez se tuvo la idea de partida, se comenzó a escribir el código que daría respuesta al problema. El mismo se compone de ocho clases que se relacionan de la siguiente manera:

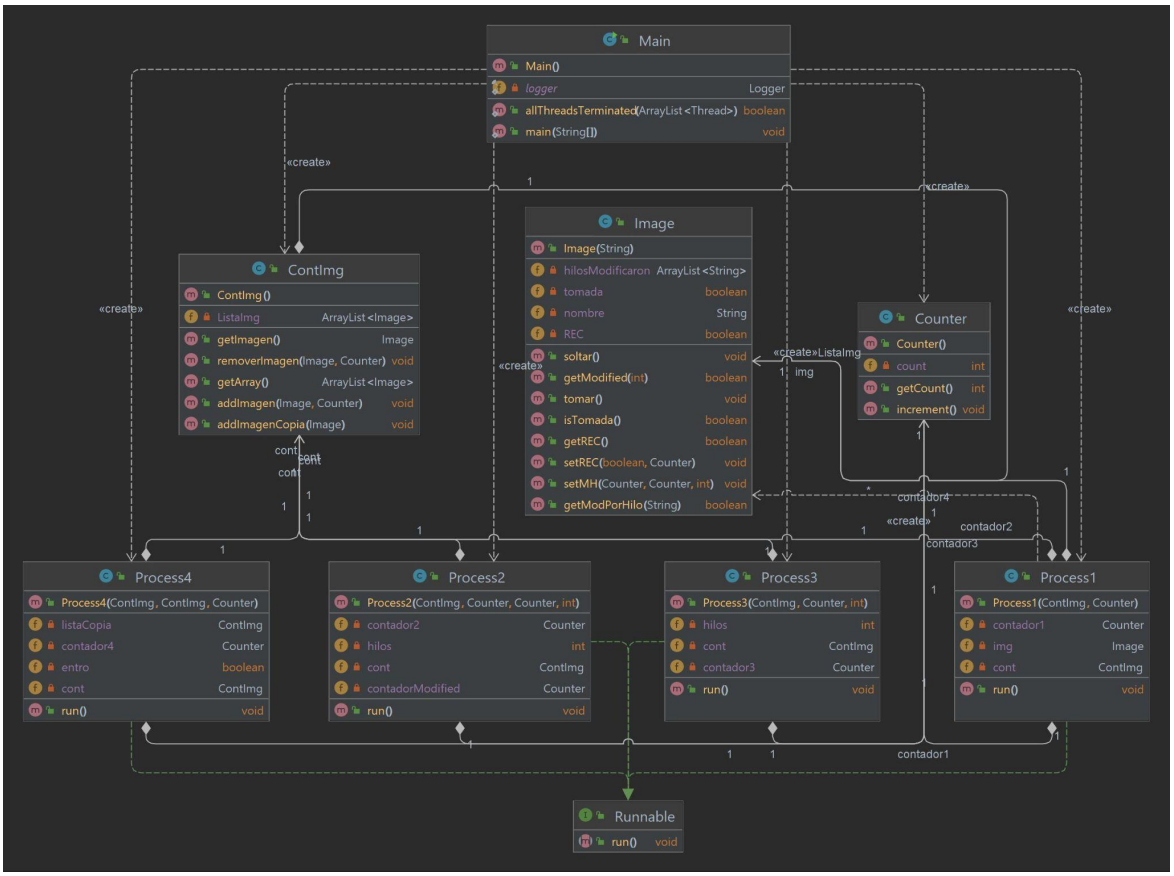


Imagen 1: diagrama de clases

Este código implementa un sistema para modificar y manipular imágenes, representadas por la clase ***Image***, donde la clase ***ContImg*** sirve como contenedor para las imágenes y proporciona métodos para agregarlas y eliminarlas del contenedor. Este proceso lo hace implementando un ***ArrayList*** de ***Image***; también posee métodos ***addImagen, addImagenCopia***, ***getImagen***, ***getArray*** y ***removerImagen,*** destinados a las operaciones básicas a realizar sobre el banco de imágenes, todos estos métodos constan de bloques ***synchronized*** para evitar que los hilos de los procesos realicen consultas o modifiquen el banco al mismo tiempo; aquí cabe aclarar que el método ***getImagen*** es el más importante con respecto a la sincronización debido a que este debe asegurar que cada imagen pueda ser tomada por un solo hilo a la vez.

La clase ***Counter*** se utiliza para realizar un seguimiento del número de imágenes que se han modificado o eliminado del contenedor. Para llevarlo a cabo implementa una variable llamada ***count*** y dos métodos ***synchronized*** llamados ***increment*** y ***getCount*** para modificarla y obtenerla respectivamente.

La clase ***Process1*** cuenta con tres atributos ***img, cont*** que es el contenedor de imágenes y proviene de la clase ***ContImg*** y ***contador1*** que se utilizó para llevar la cuenta de cuantas fotos van entrando al array, con el fin de saber si se llegó a la

cantidad prevista (100) y para el ***log***. También sobrescribe el método ***run*** para añadir imágenes al banco.

Process2 consta de tres atributos: ***cont,*** que al igual que en el proceso anterior es el contenedor de las imágenes de la clase ***ContImg***; dos contadores: ***contador2*** que llevan la cuenta de la cantidad de veces que un hilo modificó una imagen (por lo que debe llegar a (cantidadDeHilosProceso2)*100) y ***contadorModified*** que lleva la cuenta de las imágenes que fueron modificadas por todos los hilos de este proceso. El método ***run*** de este proceso simula la modificación de las imágenes, tomando una imagen aleatoria del contenedor e implementando el método propio de la imagen.

Process3 tiene un atributo nuevo: ***contador3***, el cual sirve para llevar la cuenta de cuántas imágenes han sido recortadas. Su método ***run*** toma una imagen del banco y verifica si ésta ya ha sido recortada previamente, y en caso de no ser así, entonces usa ***setRec*** para simular el recorte modificando la flag en true y aumentando el contador.

Process4 emplea dos atributos para simular la finalización de la edición: ***contador4*** se usa para llevar el número de imágenes que fueron removidas del contenedor***, entro*** se utiliza para entrar al ciclo ***while*** cuando el contenedor principal está vacío al inicio de la ejecución, evitando que el proceso finalice en ese momento; luego, cuando se comprueba que el contenedor ya no está vacío se pone en ***false*** para que en los restantes ciclos la única comprobación sea respecto a si está vacío o no el contenedor.

La clase ***Image*** modela las imágenes que deben ser procesadas. Esta clase tiene dos campos booleanos ***REC*** y ***tomada*** para indicar si la imagen fue o no recortada y para saber si la imagen está o no tomada, respectivamente; y también cuenta con un ArrayList de Strings donde se almacenan los nombres de los hilos que modificaron a la imagen. Los métodos ***tomar*** y ***soltar*** se utilizan para indicar cuándo se está modificando la imagen y garantizar que sólo un hilo está modificando la imagen.

Por último, pero no menos importante, la clase ***main*** es la encargada de organizar los procesos que lleva a cabo el programa. Esta crea los contadores de la clase ***Counter*** que se utilizan para llevar un registro de la cantidad de imágenes que han pasado por cada uno de los procesos y la cantidad de imágenes que han sido completamente modificadas. Aquí también se crea un objeto ***cont*** y un objeto ***listaCopia*** de la clase ***ContImg***, uno es el contenedor de imágenes que se utiliza para transferir las imágenes entre los diferentes procesos y el otro es el contenedor al que se le transfieren las imágenes totalmente procesadas. Luego se implanta un escáner para pedir por consola la cantidad de hilos a asignar a cada proceso. Seguido de esto, se crean los hilos y los procesos de manera cíclica según las cantidades antes pedidas. Cada proceso es una tarea que se ejecuta en paralelo utilizando un hilo diferente. Finalmente, se emplea un ***Logger*** para escribir el log del programa, que muestra información relevante sobre el estado del programa en

tiempo real. El log se escribe cada 500 milisegundos. El programa se detiene cuando los cuatro procesos terminan de tratar todas las imágenes. Al final se muestra el tiempo de ejecución total del programa.

Se puede observar en el siguiente diagrama de secuencias las interacciones dentro del programa:

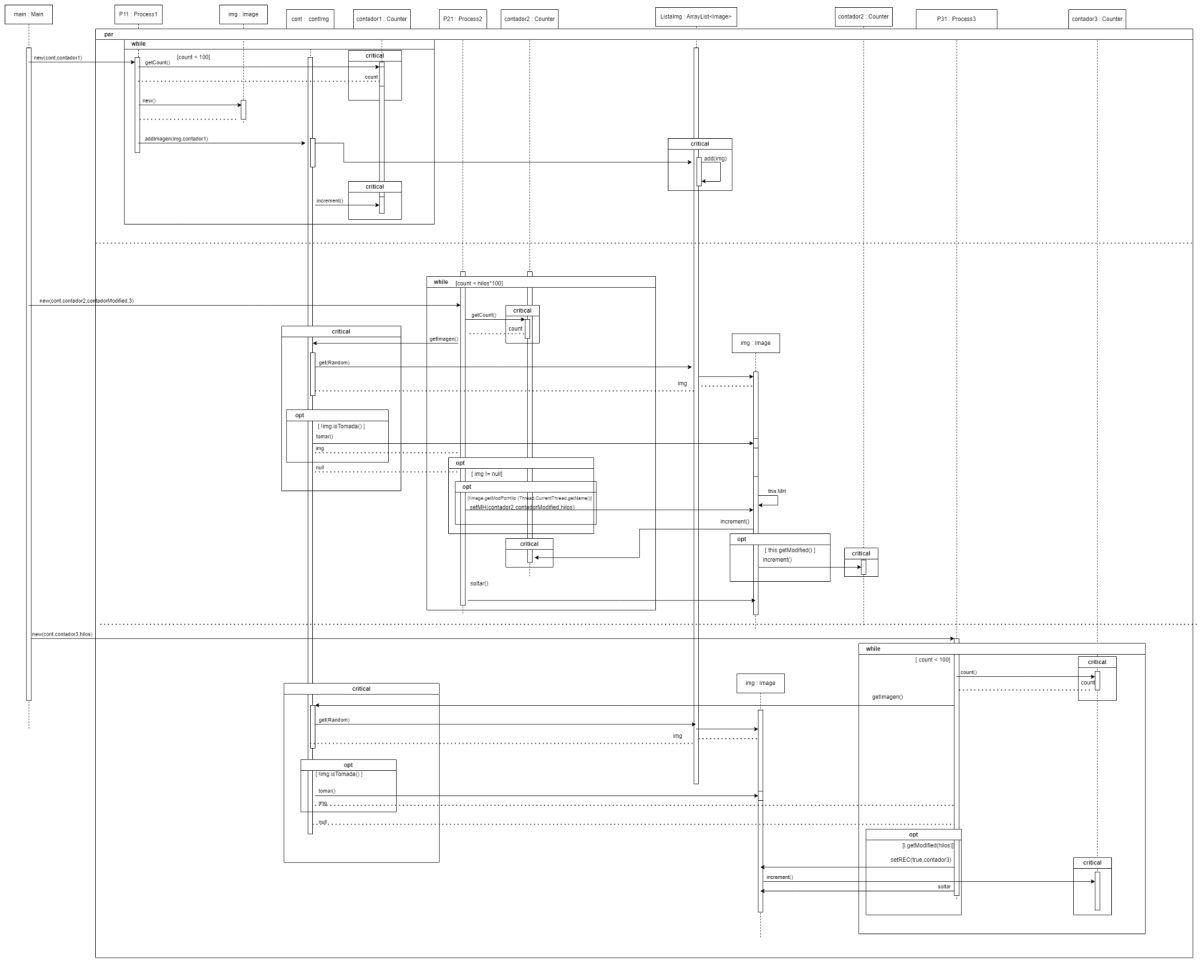


Imagen 2: diagrama de secuencias

Junto con el informe se adjuntan los archivos originales de las imágenes 1 y 2 en calidad superior.

Respecto a los tiempos, a continuación se encuentran tabuladas las demoras que se le asignó a cada proceso:

Proceso	Demora asignada
Carga de imágenes al contenedor	10 ms cada hilo por imagen.
Mejora de iluminación de las imágenes	100 ms cada hilo por imagen, cada hilo procesa todas las imágenes.
Recorte de las imágenes	15 ms cada hilo por imagen.

Transferencia de un contenedor a otro	15 ms cada hilo por imagen.

Se realizaron 30 ejecuciones, y a partir de ello se obtuvo un promedio de 10,532 segundos de demora total. (En un sistema con un Ryzen 5 1600 de 6 cores y 12 hilos)

La elección de los tiempos se dio debido a que el proceso 2 escala prácticamente 1:1 con el tiempo del programa (Ejemplo: si se agregan 2 segundos al proceso 2, se suman casi 2 segundos al programa final)

Esta situación se debe a que al necesitar todos los hilos el proceso 2, se asegura que el tiempo puesto en este proceso va a ocurrir si o si. Por lo que se utilizó este proceso como pivote para asegurar más de 10 segundos de ejecución.

El resto de procesos se mantienen con un tiempo muy corto, aportando solo 0,5 segundos aproximadamente al tiempo total (Cuando se respetan los hilos de la consigna), ya que cuando se cargaron 10 imágenes recién se está terminando de modificar una, y cuando una imagen es modificada por completo, casi instantáneamente es recortada, copiada y eliminada. De esta forma, con los procesos 3 y 4 se observan claramente los beneficios de la concurrencia, además se considera que esto se corresponde con la realidad, ya que la edición de una imagen toma mucho más tiempo que cualquier otro de los procesos solicitados. Lamentablemente, la decisión de usar como pivote el proceso 2 no permite ver correctamente la concurrencia en el log durante la carga de imágenes, sin embargo cambiando los tiempos de ejecución se puede observar el correcto paralelismo del programa también en este punto de la ejecución.

CONSIDERACIONES

Entre las decisiones y consideraciones de diseño más relevantes para el grupo, una de las instancias que más debate produjo fue acerca de cómo implementar y plasmar en el código el segundo proceso, donde el requerimiento de que cada imagen fuera tratada por todos los hilos correspondientes sin bloquear en ningún momento a los mismos resultó un desafío para los integrantes. Para resolverlo, el grupo implementó como solución para este y otros casos donde la concurrencia podría significar un problema, la flag ***tomada*** y la palabra clave ***synchronized***. La primera es una variable que se utiliza para indicar el estado de una imagen como ocupada, y la segunda bloquea el acceso simultáneo de varios hilos a un recurso compartido, asegurando que solo un hilo pueda acceder al recurso en un momento dado.

Una de las consideraciones más importantes a tener en cuenta por el grupo, fue concebir un código de fácil lectura e interpretación. Para ello se tomaron medidas como documentar los distintos métodos y demás componentes del código, emplear nombres claros para cada variable, clase y método que sean

representativos para los mismos, y se buscó respetar las buenas prácticas de programación; dando así lugar a una buena comprensión para quien lea el código fuente del programa.

CONCLUSIÓN

Este trabajo fue el primero de varios proyectos vinientes donde será necesario implementar la programación concurrente. Esto le permitió al grupo comprender en primera mano las ventajas de la misma, entre ellas se observaron un mejor aprovechamiento de los recursos de hardware, mayor rapidez al permitir que los procesos puedan llevarse a cabo en simultáneo y también la posibilidad de acelerar la capacidad de respuesta al incrementarse recursos de hardware.

Para finalizar, el grupo concluye en la utilidad que representó el presente trabajo práctico para afianzar los contenidos aprendidos en clase, recordar conceptos aprendidos en asignaturas anteriores y experimentar los desafíos que la programación concurrente impone.

A pesar de las dificultades que el proyecto significaba no sólo por sí mismo sino también por ser el primero de la asignatura, el grupo logró poner en práctica los recursos ofrecidos por la cátedra, pudiendo así resolver la consigna cumpliendo con los objetivos detallados al inicio del informe.

BIBLIOGRAFÍA

- Diapositivas de clase.
- Instructivos y filminas facilitadas por el docente.