Angie Ximena López Cruz - 202312848

Juan Diego Rodriguez Barragan - 202221822

Carlos Felipe Vargas Morales -202220064

Documentación Caso 2

- **Diagrama de clases**

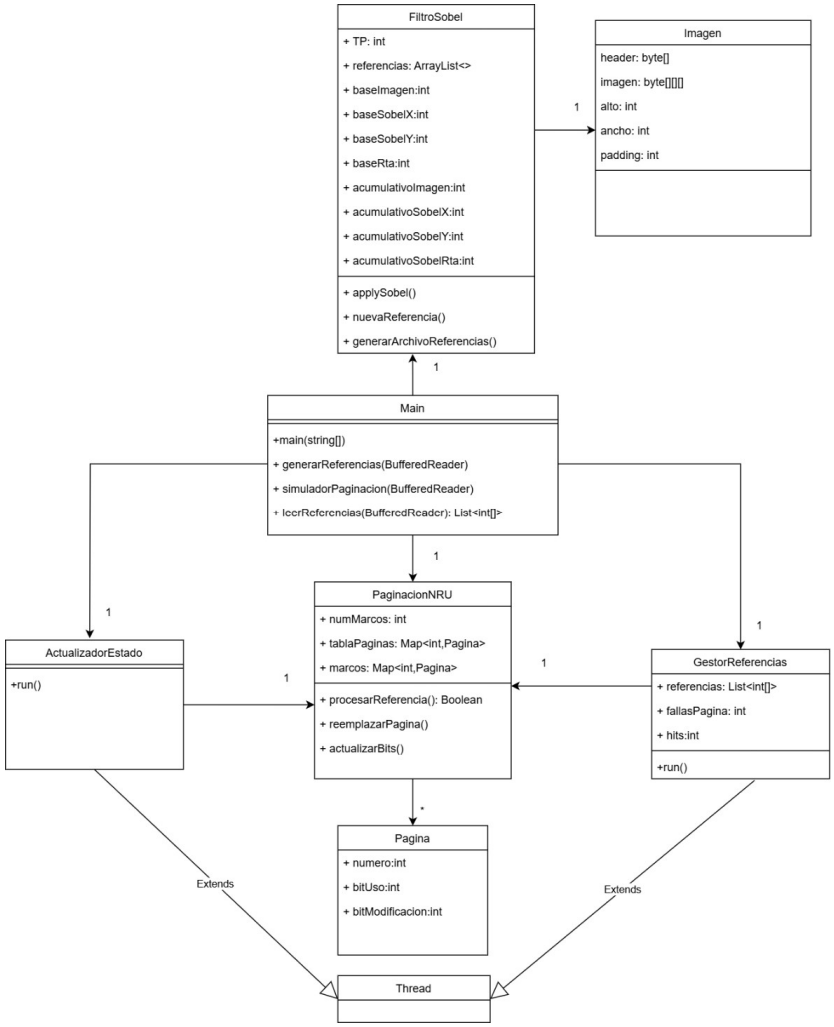

Imagen 1. Modelo de clases para la implementación del sistema de paginación.

El diagrama representa una simulación del algoritmo de paginación NRU. La clase **Main** coordina la ejecución general; **FiltroSobel** genera las referencias de acceso a páginas a partir del procesamiento de imágenes; **Imagen** se encarga de leer y escribir imágenes BMP; **PaginacionNRU** gestiona los marcos, el reemplazo de páginas y los bits de uso/modificación; **Pagina** modela una página individual; **GestorReferencias** lee la secuencia de referencias y contabiliza hits y fallos; **ActualizadorEstado** reinicia periódicamente los bits de uso. Tanto **GestorReferencias** como **ActualizadorEstado** extienden de **Thread**, permitiendo ejecución concurrente durante la simulación.

Angie Ximena López Cruz - 202312848

Juan Diego Rodriguez Barragan - 202221822

Carlos Felipe Vargas Morales -202220064

- **Descripción del algoritmo usado para generar las referencias de página (Opción 1)**

- **Objetivo:**

El objetivo fue transformar cada acceso de memoria lógico (lectura o acceso y mascara o resultado) en una referencia de página virtual que se escribe en un documento para posteriormente ser leída para la opción 2. Esto con el fin de simular como el sistema operativo manejaría el acceso en memoria paginada (page mapping).

- **Contexto:**

Para entender nuestro algoritmo primero es importante entender un poco lo que hace el algoritmo de sobel. Este es un algoritmo de procesamiento de imágenes que permite detectar sus bordes por medio de cambios abruptos de intensidad en una imagen. Este tiene 4 componentes:

- ImagenIn: Fuente de datos originales sobre la que se hace el procesamiento
- SOBEL_X: Matriz de gradiente horizontal (detecta bordes verticales)
- SOBEL_Y: Matriz de gradiente vertical (detecta bordes horizontales)
- ImagenOut :Donde se guarda el resultado de gradiente

- **Asignación de bases lógicas:**

Para lograr la opción 1 primero asignamos una base lógica acumulativa a cada componente:

Componente	Base de Dirección Virtual	Tamaño estimado
ImagenIn	0	Alto*Ancho*3bytes
Sobel	Después de ImagenIn	36 bytes (3*3*4)
Sobel	Después de ImagenIn	36 bytes (3*3*4)
ImagenOut	Después de ImagenIn	Alto*Ancho*3bytes

Las bases de dirección virtual se calculan de manera acumulativa de manera que se simule como un sistema operativo asigna espacios contiguos en memoria virtual a los datos de un proceso.

1. La imagenIn comienza en la dirección virtual 0
2. Luego SOBEL_X y SOBEL_Y que ocupan 36 bytes cada una se colocan de manera contigua primero SOBEL_X y después SOBEL_Y.g
3. Por último, imagenOut se ubica después de los filtros.

Las bases serán usadas para calcular las direcciones virtuales absolutas de cada elemento accedido aplicando la fórmula:

Formula:	dirección_virtual = base + desplazamiento

Angie Ximena López Cruz - 202312848

Juan Diego Rodriguez Barragan - 202221822

Carlos Felipe Vargas Morales -202220064

- **Generación de referencia:**

Posteriormente generamos las referencias. Cada vez que el código accede a

- Un color de un píxel de entrada imagenIn este tiene 3 referencias de lectura(R) una por cada canal RGB
- Un valor de una máscara SOBEL_X o SOBEL_Y este tiene 3 referencias de lectura(R) por el tipo int
- Un píxel de la imagen de salida imagenOut este tiene 3 referencias de escritura(W) una por cada canal RGB

De manera que se cree una referencia siguiendo este formato:

Formato	String ref = nombre + "," + pagina + "," + desplazamiento + "," + bitAccion;
Donde:	• Nombre: nombre de la matriz y índice de su posición especifica • Pagina=dirección/TP • Desplazamiento=direccion%TP • BitAccion=R(lectura) o W(Escritura)

Cada una de las referencias se guardan en una lista para posteriormente ser escritas en un .txt

- **Cálculo de metadatos:**

A continuación, calculamos los metadatos antes de escribir las referencias de esta manera:

TP: Tamaño de página la cual es dada por el usuario

NF: Numero de Filas de la imagen

NC: Numero de Columnas de la imagen

NR: Tamano de la lista de referencias una vez procesada la imagen

NP:

int totalBytes = (alto * ancho * 3 * 2) + 72.

- El alto y ancho son las dimensiones de la imagen en pixeles.
- *3: Cada píxel tiene 3 bytes uno por cada canal RGB.
- *2 por las 2 imágenes imagenIn y imagenOut
- +72 ya que cada filtro sobel ocupan una matriz 3x3 de 4 bytes es decir 3*3*4=36 y como es SOBEL_X y SOBEL_Y entonces 36*2=72

int NP = Math.ceil(totalBytes / TP).

Esta fórmula nos permite calcular el número de páginas necesarias y en caso de que el cálculo quede en decimales Math.ceil se asegura de que incluso si sobra espacio en la última página esta se cuente.

- **Escritura del Archivo de referencias:**

  Por último, guardamos el archivo .txt dentro de la carpeta /Referencias siguiendo el formato que se pide incluyendo los metadatos y las referencias generada

- **Descripción de las estructuras de datos usadas para simular el comportamiento del sistema de paginación y cómo usa dichas estructuras (cuándo se actualizan, con base en qué y en qué consiste la actualización)**

Para simular el comportamiento del sistema de Paginación usamos las siguientes estructuras en el algoritmo NRU, implementado en la clase PaginacionNRU:

1. **Tabla de páginas:** Map<Integer, Pagina> tablaPaginas

   Esta estructura simula la tabla de páginas, donde se registran las páginas referenciadas. Por simplicidad, se actualiza cada vez que una página es referenciada, se agrega a esta tabla, y cuando deja de estar en la memoria RAM, se elimina.
   Su principal función es verificar si una página se encuentra en un marco de la memoria RAM. Si no está presente, se genera un fallo de página y, si es necesario, se aplica el algoritmo NRU para seleccionar una página a reemplazar

2. **Marcos de página:** LinkedHashMap<Integer, Pagina> marcos

   Esta estructura representa los marcos de página en la memoria RAM, donde se almacenan las páginas actualmente cargadas. Se implementa como un mapa en el que se agregan las páginas referenciadas, con la restricción de no superar el número máximo de marcos disponibles en la memoria.

   El proceso de actualización sigue tres casos: El primero si la página ya está presente en un marco, solo se actualizan los bits de uso y/o modificación, sin necesidad de realizar cambios en la estructura. El segundo es si la página no está en la memoria, pero aún hay marcos libres, ahí se agrega directamente la página a la estructura. Finalmente, el tercer caso es cuando no se encuentra referenciada y todos los marcos de página están usados, en este caso se genera un fallo de página y se aplica el algoritmo NRU para reemplazar la página, por lo que se actualiza en la estructura.

3. **Página:** Clase Pagina

Esta clase simula una Pagina virtual, donde se almacena su numero de página, su bit de uso y bit de modificación que pueden ser 0 o 1. Para su actualización se hace cuando se crea, ahí inicia con su bit de uso en 1 y depende si hubo escritura se actualiza el de modificación. Esta clase también se modifica cuando el thread de actualizador de estado modifica cada 1 milisegundo el bit de uso de las páginas que están en el marco de referencia.

4. **Lista de referencias:** List<int[]> referencias

Esta estructura representa la secuencia de referencias a memoria donde contiene todas las referencias de páginas a ser procesadas, cada elemento [numeroPagina, esEscritura] indica qué página es referenciada y si es una operación de escritura.
Se recorre en GestorReferencias.run() para simular accesos a memoria.

- **Esquema de sincronización usado:**

Como en la simulación se involucran dos hilos, uno que gestiona las referencias done se simula el acceso a memoria virtual (GestorReferencias) y otro que se encarga de cambiar el estado de usado de una página (ActualizadorEstado), es necesario controlar el acceso a las estructuras compartidas mediante una sincronización que permite que no haya inconsistencia en los datos que maneja PaginacionNRU.

La sincronización se usa en todos los métodos de la clase PaginacionNRU, es decir en:

1. **procesarReferencia(int numeroPagina, boolean esEscritura**):

Porque este método es llamado por el hilo GestorReferencias donde actualiza las estructuras marcos y tablaPaginas que las usa los métodos llamados por el otro hilo. Así que para mantener la consistencia es necesario evitar condiciones de carrera en los cambios de las estructuras permitiendo que solo uno a la vez pueda modificarlos.

2. **reemplazarPagina():**

Porque este método modifica marcos y tablaPaginas, eliminado paginas de memoria, por lo que debe de asegurarse de que el otro hilo no intente leer páginas que ya han sido eliminadas de memoria o eliminar una sin considerar la última que acaba de ser agregada.

3. **actualizarBits():**

Porque este método modifica los bits de uso de las páginas referenciadas en los marcos de página. Es necesario que no se ejecute simultáneamente con reemplazarPagina(), para evitar inconsistencias al leer los bits al momento de seleccionar qué página eliminar o intentar acceder a una página que ya fue eliminada.

- **Tabla con los datos recopilados**

Angie Ximena López Cruz - 202312848

Juan Diego Rodriguez Barragan - 202221822

Carlos Felipe Vargas Morales -202220064

parrotspeq.bmp			
119 x 79 px			
Páginas de 512B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	756756	595037	161719
4	756756	742254	14502
6	756756	756646	110
8	756756	756646	110
Páginas de 1024B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	756756	596878	159878
4	756756	756548	208
6	756756	756701	55
8	756756	756701	55
Páginas de 2048B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	756756	602637	154119
4	756756	756728	28
6	756756	756728	28
8	756756	756728	28
Páginas de4096B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	756756	614521	142235
4	756756	756742	14
6	756756	756742	14
8	756756	756742	14

animals.bmp			
120 x 80 px			
Páginas de 512B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	773136	607911	165225
4	773136	752419	20717
6	773136	773020	116
8	773136	773022	114
Páginas de 1024B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	773136	610651	162485
4	773136	770344	2792
6	773136	773080	56
8	773136	773080	56
Páginas de 2048B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	773136	616821	156315
4	773136	771891	1245
6	773136	773108	28
8	773136	773108	28
Páginas de4096B			
Marcos Asignados	**Total referencias**	**Hits**	**Fallas**
2	773136	628924	144212
4	773136	772598	538
6	773136	773122	14
8	773136	773122	14

Imagen 2. Registro de fallos de página y hits según el tamaño de página y el número de marcos.

Se presentan **dos escenarios principales** de simulación (parrotseq.bmp y animals.bmp). Para cada uno se recopilaron datos variando **cuatro tamaños de página** (512B, 1024B, 2048B y 4096B) y **cuatro cantidades de marcos asignados** (2, 4, 6 y 8). En cada combinación se registraron el total de referencias, los hits y los fallos de página.

- **Gráficas que ilustran el comportamiento del sistema**

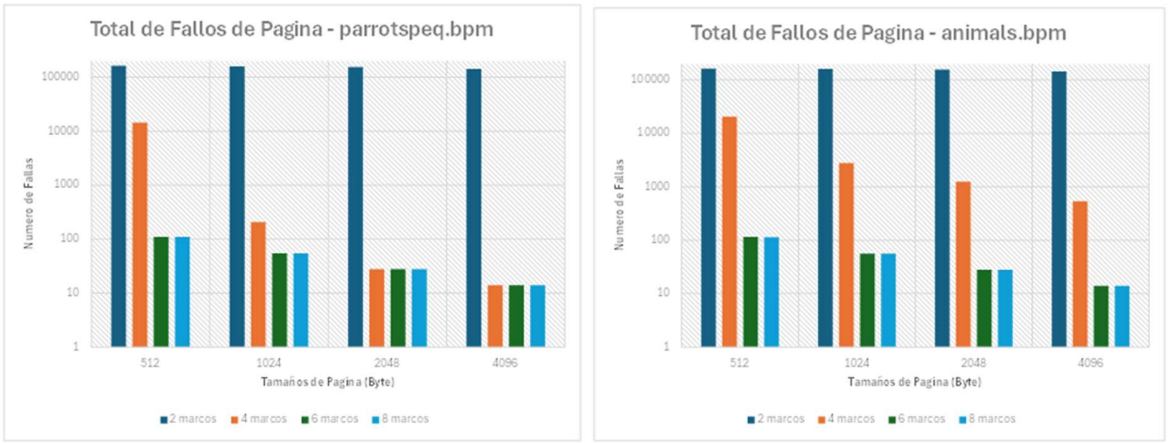

Imagen 3. Comportamiento de los fallos de página en función del tamaño de página y el número de marcos.

Angie Ximena López Cruz - 202312848

Juan Diego Rodriguez Barragan - 202221822

Carlos Felipe Vargas Morales -202220064

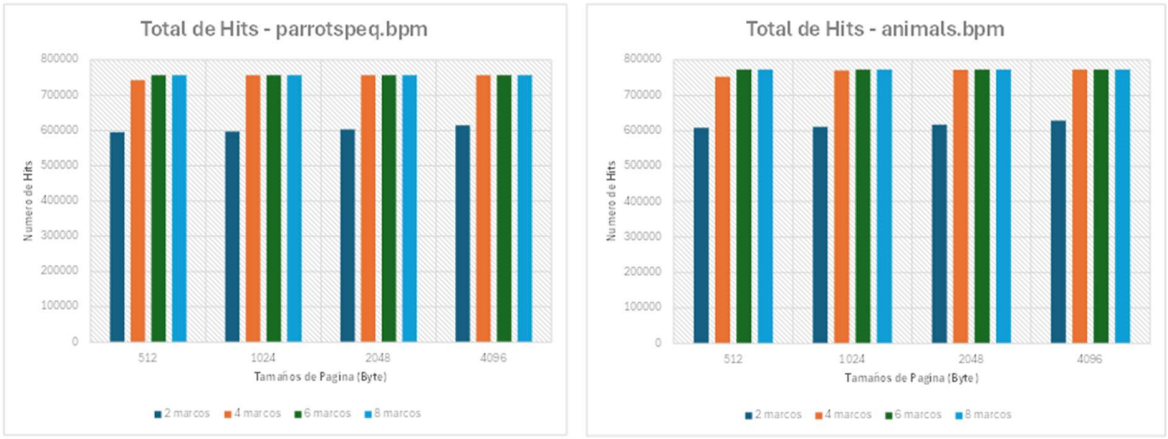

Imagen 4. Comportamiento de los hits en función del tamaño de página y el número de marcos.

- **Gráficas de tiempo**

parrotspeq.bmp				
Tiempo Ejecucion				
Tamaño de Página (B)	**2 marcos**	**4 marcos**	**6 marcos**	**8 marcos**
512	380,747	342,596	317,927	330,097
1024	382,168	326,409	310,793	343,725
2048	388,455	309,947	288,324	308,499
4096	355,588	316,213	322,436	333,043

animals.bmp				
Tiempo Ejecucion				
Tamaño de Página (B)	**2 marcos**	**4 marcos**	**6 marcos**	**8 marcos**
512	400,919	351,015	350,408	316,439
1024	331,798	329,601	326,471	326,241
2048	359,186	326,946	334,756	380,747
4096	370,030	336,228	327,146	341,038

Imagen 5. Registro del tiempo de ejecución según el tamaño de página y el número de marcos.

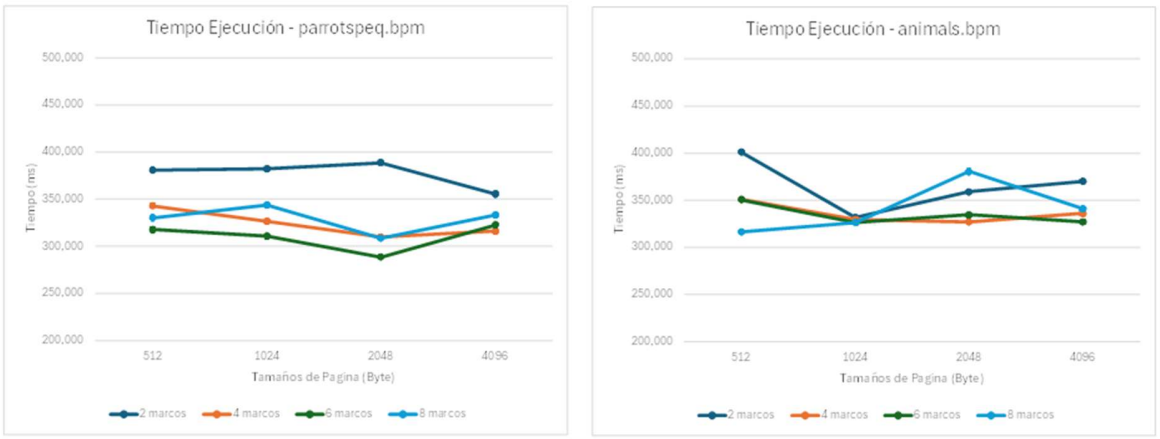

Imagen 6. Comportamiento del tiempo de ejecución en función del tamaño de página y el número de marcos.

- **Interpretación de resultados**

Angie Ximena López Cruz - 202312848

Juan Diego Rodriguez Barragan - 202221822

Carlos Felipe Vargas Morales -202220064

El análisis de los resultados evidencia que **los fallos de página son inversamente proporcionales** al tamaño de las páginas y al número de marcos de página asignados, mientras que **los hits muestran una relación directamente proporcional** con estas variables. Esta relación se explica porque, al incrementar el tamaño de las páginas, es decir, el número de bytes que cada página puede almacenar, se necesita un menor número de páginas para contener la totalidad de la información de la imagen, lo que reduce la probabilidad de que una página requerida no esté en memoria. Por otro lado, a mayor cantidad de marcos de página disponibles, más páginas pueden mantenerse en RAM simultáneamente, disminuyendo así los fallos de página y la necesidad de ejecutar el algoritmo NRU (páginas no usadas recientemente) para reemplazos. En cuanto **al tiempo de ejecución, también se observa una relación inversamente proporcional** respecto al tamaño de página y número de marcos, pues al reducirse la cantidad de fallos y la frecuencia de ejecución del algoritmo, se optimiza el rendimiento general del sistema.

- **¿Aplicar el filtro sobel representa un problema de localidad alta, media o baja?**

Consideramos que el filtro de sobel tiene una localidad alta ya que este recorre sistemáticamente píxel por píxel, revisando una ventana 3x3 de vecinos en cada iteración. Es por eso por lo que se generan accesos consecutivos a memoria tanto en espacio como en tiempo. En cuanto a memoria Espacial se accede a la ventana 3x3 de cada píxel y se accede a las mismas posiciones de las mascara de SOBEL_X y SOBEL_Y es decir que estas se ubican en una porción contigua y fija en memoria (acceso localizado).

Por el lado de memoria Temporal las mascara de SOBEL_X y SOBEL_Y son accedidas repetida mente esto implica que si se mantuvieran en RAM se evitarían muchos fallos de página. Así mismo los pixeles de imagenIn también son reutilizados ya que la ventana 3x3 desliza la imagen. Obligando que los mismos pixeles sean utilizados en pasos seguidos aumentando la reutilización temporal.