Caso 2 – Estudio de Memoria Virtual

Nombres:

David Mora Ramirez - 202226269

Julian Parra - 202013033

Descripción del algoritmo de la opción 1

El algoritmo se encarga de generar un archivo de referencias a partir de la ejecución del método *applySobel* aplicado a una imagen BMP. Los pasos son los siguientes:

- **Carga de la Imagen:**
  Se lee la imagen BMP usando la clase Imagen, obteniendo su ancho, alto y calculando el padding correspondiente. La imagen se almacena en una matriz tridimensional, donde cada píxel tiene 3 componentes (B, G, R).

- **Cálculo de Espacios en Memoria:**
  Se calculan los tamaños en bytes de las siguientes estructuras:

  Imagen original: Se utiliza el número de píxeles (alto × ancho) multiplicado por 3.

  Filtros Sobel (SOBEL_X y SOBEL_Y): Cada filtro es una matriz 3×3 de enteros (4 bytes por entero).

  Imagen Resultado: Similar al tamaño de la imagen original.

- **Determinación de Bases Virtuales:**
  Las matrices se organizan secuencialmente en memoria virtual:

  La imagen ocupa la primera región, seguida por el filtro SOBEL_X, luego SOBEL_Y y finalmente la imagen de salida (resultado).

- **Cálculo de Páginas Virtuales:**
  Se suma el tamaño total de las estructuras y se divide (redondeando hacia arriba) por el tamaño de página especificado. Esto determina el número total de páginas virtuales necesarias.

- **Generación de Referencias:**
  Para cada píxel *central* (se omiten los bordes) se generan referencias a:

  Imagen Original: Se generan 3 referencias (una para cada componente: r, g, b) basadas en su posición en memoria (calculando offset y página).

  Filtros SOBEL_X y SOBEL_Y: Para cada uno de los 9 elementos del kernel, se generan 3 referencias (repetidas) para simular su acceso durante el cálculo del gradiente.

- **Imagen Resultado:** Se generan referencias de escritura (W) para cada componente del píxel.

  Cada referencia contiene:

  - Identificador (por ejemplo, Imagen[2][3].r),
  - Número de página virtual,
  - Offset dentro de la página,
  - Tipo de acceso (R para lectura, W para escritura).

Descripción de la estructura de datos

- Para simular el comportamiento de la memoria virtual se emplean las siguientes estructuras:

  - **Tabla de Páginas (tablaPaginas):** Un arreglo de objetos de la clase *PaginaInfo*, donde cada objeto contiene:
    - **bitR:** Indica si la página fue referenciada.
    - **bitM:** Indica si la página fue modificada. Estos bits se actualizan en cada acceso: Al leer o escribir, se marca bitR = 1. Si la operación es de escritura, se marca también bitM = 1.

  - **Marcos de Página (marcos):** Un arreglo de enteros que representa los marcos físicos en RAM. Cada posición almacena el número de página cargado o -1 si está libre. La actualización de esta estructura ocurre en el método *acceder*:
    - Se verifica si la página requerida ya está en algún marco (hit).
    - Si no está, se produce una falla de página (miss) y se carga la página en un marco libre o, si no hay disponibles, se invoca el algoritmo de reemplazo NRU.

  - **Algoritmo de Reemplazo NRU:** Cuando se requiere reemplazar una página, se clasifica cada página presente en los marcos según el valor de sus bits (clases 0 a 3, siendo 0 la mejor candidata para reemplazo). Se selecciona la primera página de la clase más baja.

  - **Actualización Periódica de Bits:** Un hilo (hilo B) se encarga de limpiar los bits R de todas las entradas en la tabla cada 1 ms, simulando la "limpieza" periódica que se realiza en un sistema real para el algoritmo NRU.

Esquema de Sincronización

Para garantizar la coherencia en el acceso a las estructuras compartidas (*tablaPaginas* y *marcos*) se implementó un mecanismo de sincronización utilizando un objeto de bloqueo (lock):

¿Dónde se aplica la sincronización?

- En el método *acceder*, donde se actualizan los bits y se verifica la presencia de la página en los marcos.
- En el método *limpiarBitsR*, que se ejecuta en un hilo independiente y resetea los bits R de todas las páginas.

Justificación: La sincronización es necesaria para evitar condiciones de carrera entre el hilo que procesa las referencias y el hilo que limpia los bits R. Sin este mecanismo, podrían producirse inconsistencias en el estado de la tabla de páginas y errores en el conteo de hits y misses.

Datos recopilados

Tablas para la imagen parrotspeq.bmp

Con 4 marcos:

parrotspeq.bmp					
Marcos	TP	Total de referencia	Hits	Misses	Tiempo total estimado
4	32	756756	507052	249704	2,4753E+12
4	64	756756	547223	209533	2,09594E+12
4	512	756756	742869	13887	1,37887E+11
4	1024	756756	756472	284	4947813250
4	2048	756756	756720	36	397836000

Con 6 marcos:

parrotspeq.bmp					
Marcos	TP	Total de referencia	Hits	Misses	Tiempo total estimado
6	32	756756	522759	233997	2,20279E+12
6	64	756756	570267	186489	1,59733E+12
6	512	756756	756551	205	3567820150
6	1024	756756	756701	55	587835050
6	2048	756756	756728	28	317836400

Tablas para la imagen mariposa.bmp

Con 4 marcos:

mariposa.bmp					
Marcos	TP	Total de referencia	Hits	Misses	Tiempo total estimado
4	32	8859480	5954623	2904857	2,88411E+13
4	64	8859480	6456976	2402504	2,39827E+13

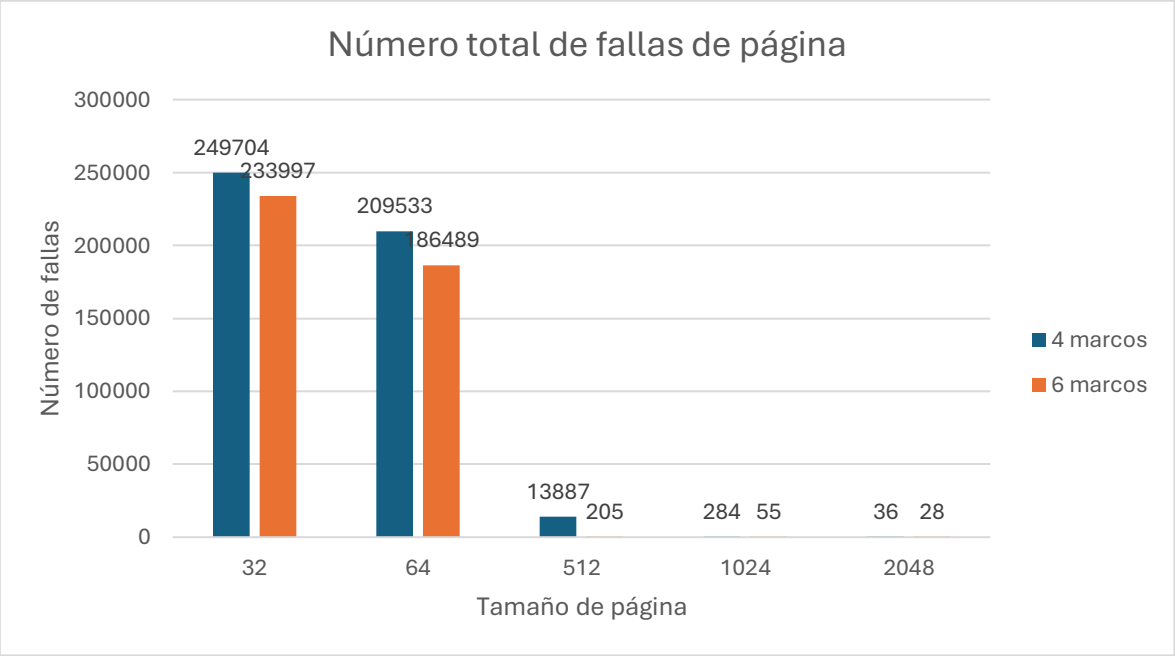
4	512	8859480	8535349	324131	2,78058E+12
4	1024	8859480	8613952	245528	2,44987E+12
4	2048	8859480	8802600	56880	5,1439E+11

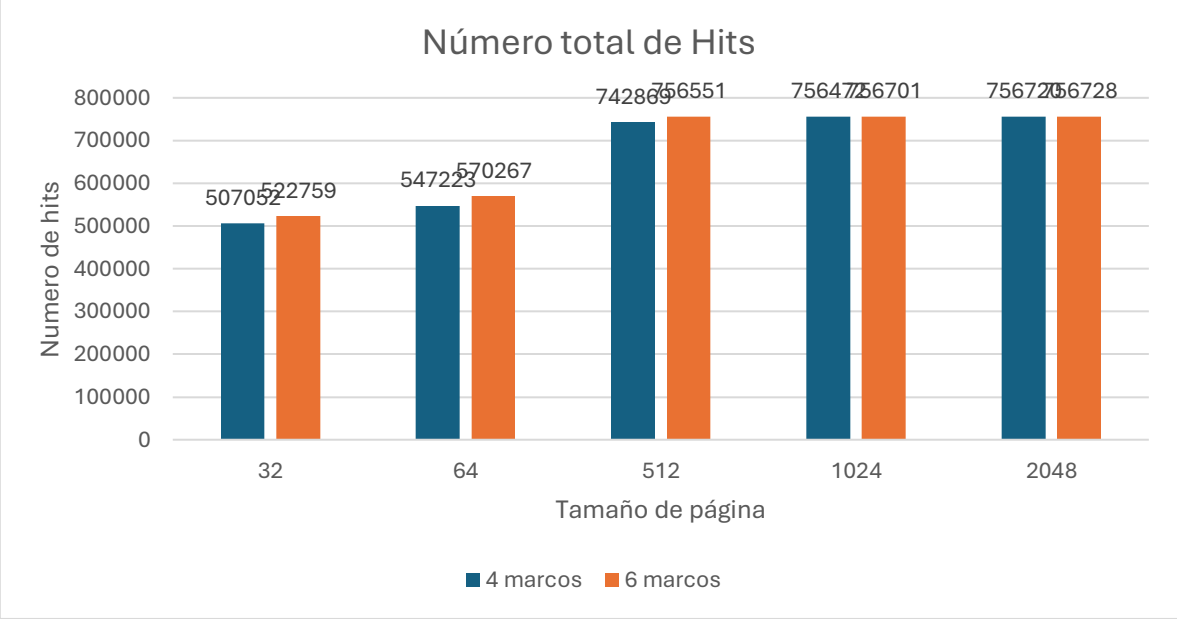
Con 6 marcos:

mariposa.bmp					
Marcos	TP	Total de referencia	Hits	Misses	Tiempo total estimado
6	32	8859480	6131324	2728156	2,73524E+13
6	64	8859480	6764598	2094882	2,03919E+13
6	512	8859480	8751344	108136	9,74128E+11
6	1024	8859480	8833362	26118	2,09872E+11
6	2048	8859480	8859167	313	3572958350

Graficas sobre comportamiento de sistema

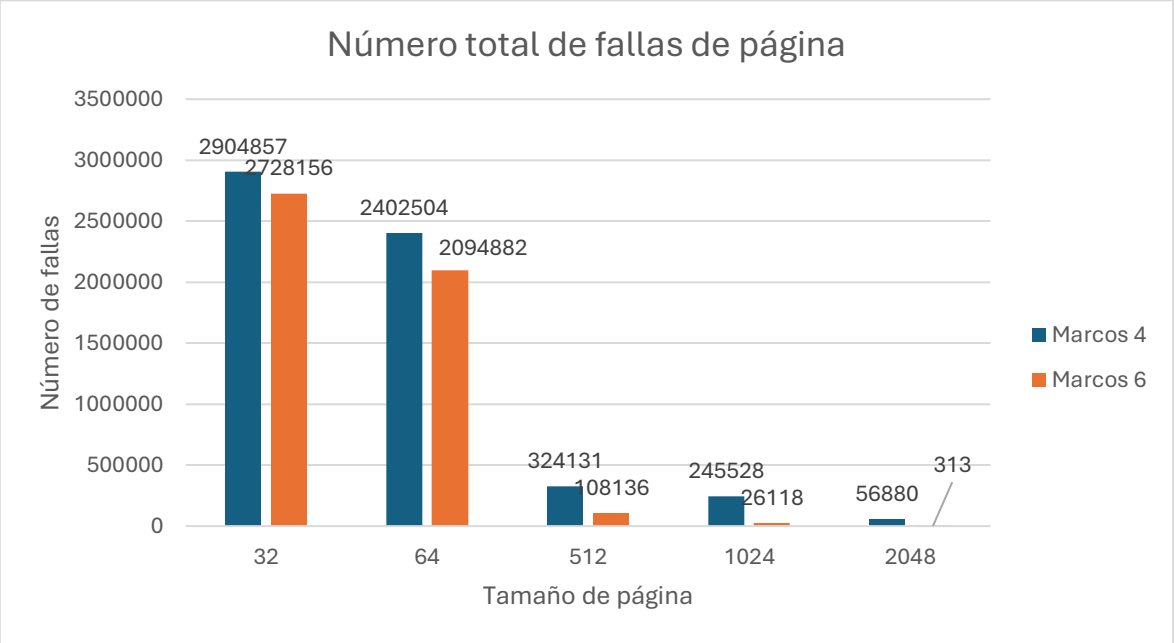
Gráficas para la imagen parrotspeq.bmp

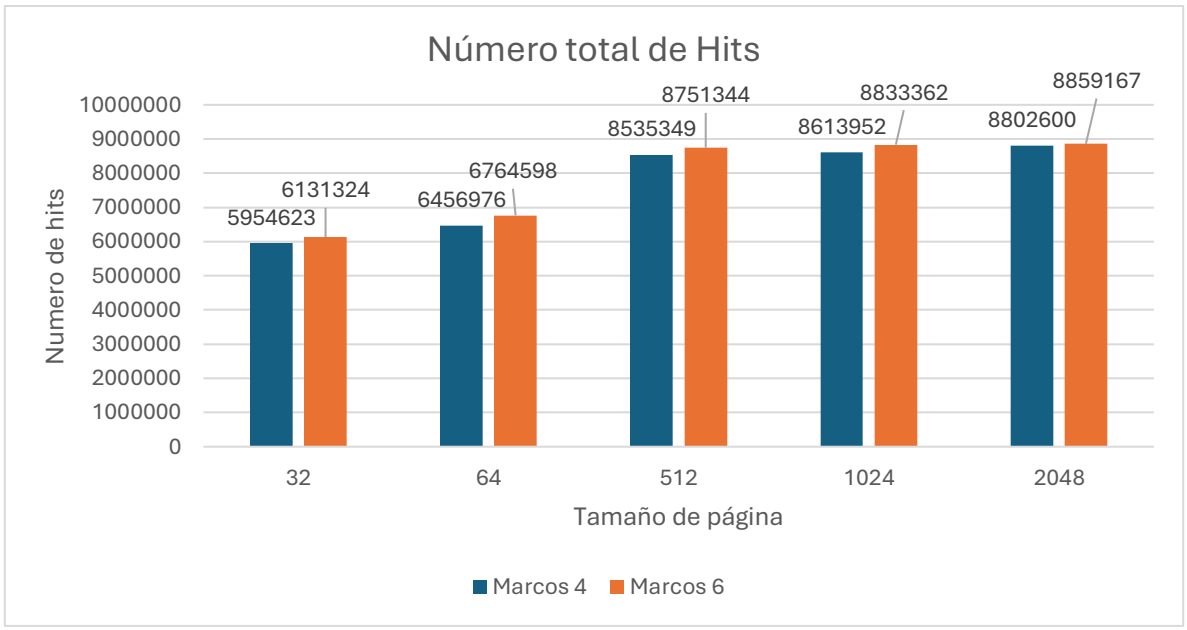




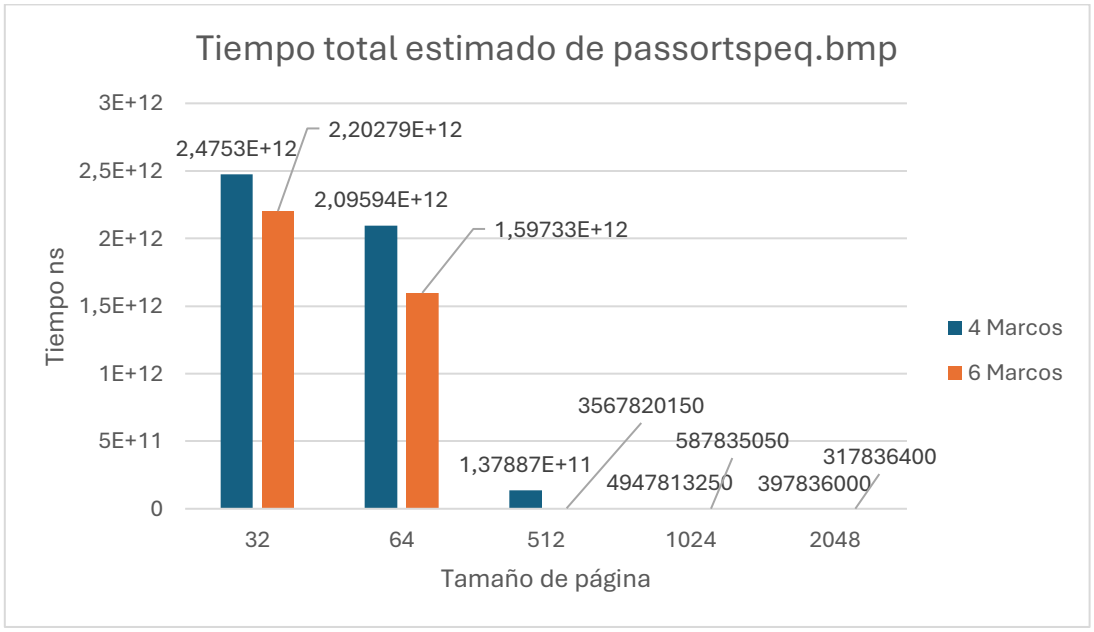
Además se propuso una imagen nueva de una mariposa (Ancho: 400 px, Alto: 267 px).

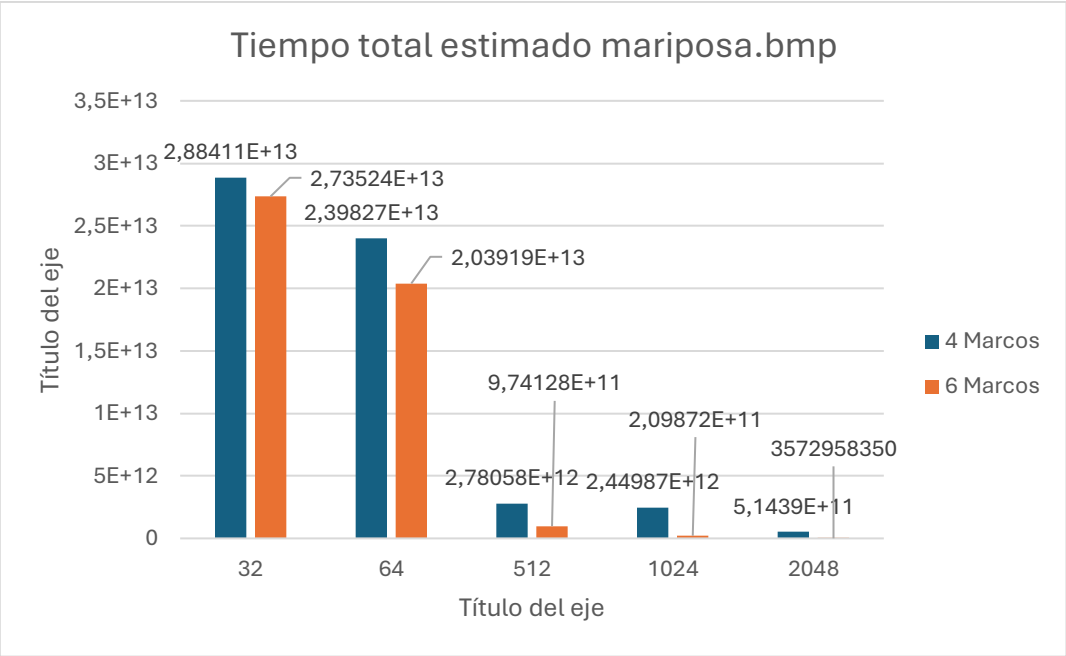
Graficas para la imagen mariposa.bmp





Graficas de tiempo





Interpretación de resultados

Los resultados obtenidos son consistentes con lo esperado: A medida que aumenta el número de marcos asignados, el número de fallas de página disminuye drásticamente y se incrementa el porcentaje de hits. Esto se debe a que disponer de más marcos permite almacenar un mayor conjunto de páginas utilizadas con frecuencia, reduciendo así los accesos a la memoria SWAP, que tienen un tiempo de latencia mucho mayor. Además, el tiempo total estimado se reduce significativamente con un mayor número de marcos, ya que el costo asociado a cada fallo de página se ve compensado por la mayor cantidad de accesos exitosos en RAM. Los resultados confirman la efectividad del algoritmo NRU en escenarios con mayor disponibilidad de marcos.

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

El procesamiento del filtro Sobel representa un problema de **localidad moderada a alta**. Esto se debe a que para cada píxel central se accede a una ventana 3x3, lo que favorece la reutilización de datos contiguos. Además, el procesamiento se realiza en orden por filas, lo que incrementa la localidad temporal al aprovechar datos que ya han sido cargados en caché. Sin embargo, el hecho de acceder a múltiples matrices (imagen de entrada, dos filtros y imagen de salida) dispersa parcialmente la localidad, ya que no todos los datos requeridos se encuentran en la misma región de memoria. En general, el diseño de acceso favorece la optimización a nivel de caché, aunque existe un compromiso entre el acceso secuencial y la dispersión de las referencias.