DOCUMENTACIÓN CASO 2 – INFRAESTRUCTURA COMPUTACIONAL

Integrantes:

- Santiago Pardo - 202013025
- Santiago Bastos - 202110792
- Santiago Ayala – 202110734

DESARROLLO

1. Descripción del algoritmo usado para generar las referencias de página (modo uno)

Para generar las referencias se utilizó la clase *GeneradorReferencias*, y se desarrolló el algoritmo en el método *calcularReferencias()*.

En primer lugar, cabe aclarar que, para hacer este cálculo de referencias, se ejecutó este método 3 veces, ya que este proceso se generalizó para añadir una sola matriz a la memoria virtual, esto teniendo en cuenta que, en una suma de matrices, todas las matrices deben tener la misma cantidad de filas y columnas, por lo cual, para añadir una matriz a la memoria virtual, se necesita únicamente la última página en la que se encuentra el último elemento de la última matriz que fue añadida, al igual que el primer desplazamiento que tendrá el primer elemento de la matriz.

Teniendo estos dos parámetros como entrada, se realiza el siguiente procedimiento para cada matriz:

En primer lugar, se revisa si el desplazamiento recibido es de 0, esto teniendo en cuenta el caso en específico en el que la matriz añadida empieza una nueva página, por lo cual, si este caso sucede, se debe añadir 1 página a la página recibida por el programa.

Posteriormente, se comparan todos los valores en un ciclo anidado con esta misma lógica, se añade iterativamente el tamaño del entero al desplazamiento, y una vez este desplazamiento alcance un número igual al tamaño de la página, nuevamente se reinicia el desplazamiento a 0, y se añade una página más a la página recibida.

En este caso específico del programa, se utilizó una matriz tridimensional para almacenar los valores de página y desplazamiento de la componente de la matriz, con el propósito de almacenar este valor para la futura impresión de los datos.

Para insertar las 3 matrices dentro de la memoria virtual, se hizo el respectivo cálculo de los índices para cada matriz, que fueron los siguientes, en términos de los valores de entrada del programa (NF, NC, TE, TP, MP)

Para la matriz A, como inicializa la página virtual, se insertaron los valores de página -1 (teniendo en cuenta el caso en el que comienza una página) y desplazamiento 0 (desplazamiento inicial)

Para la matriz B, se utilizaron las siguientes fórmulas:

- PaginaB = (NF*NC-1)/(TP/TE), esto dado que únicamente se ha insertado una matriz en todo el proceso, y el programa revisa cuántas páginas son necesarias para cubrir toda esta matriz.

- DesplazamientoB = ((NF*NC-1)%(TP/TE))*TP+TE)%TP, esto teniendo en cuenta que de esta forma se obtiene el último índice del último elemento de la matriz A, y se añade un TE para representar el siguiente desplazamiento, por último, se hace otro módulo con TP para que el resultado no se salga de la margen de un tamaño de la página

Para la matriz C, se utilizaron las siguientes fórmulas:

- ❖ PaginaC = 2*(NF*NC-1)/(TP/TE), debido a que ya se han añadido dos matrices a la página de la misma cantidad de filas y columnas, por lo que se encontró el número necesario de páginas para cubrir todos los elementos de las 2 matrices.

- ❖ DesplazamientoC = ((NF*NC-1)%(TP/TE))*TP+ DesplazamientoB + TE) % TP, es muy similar al procedimiento para el desplazamiento B, solo que ahora se tiene en cuenta el desplazamiento de la matriz B para tomar una referencia del desplazamiento.

2. 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).

Las estructuras que simulamos en este proyecto fueron: Memoria virtual, Tabla de páginas y la Memoria real con marcos de página. A continuación, vamos a explicar cómo se modelaron y como se actualizan.

- Memoria Virtual: La memoria virtual se simulo con el uso de un HashMap. Este HashMap está compuesto de una llave que es un Integer que hace referencia a la página y un ArrayList que contiene la lista de celdas de la matriz que están asociadas a esa página.

    - Actualización: La memoria virtual no se actualiza y no cambia ya que esta contiene toda la información disponible y el espacio no es un problema.

- Tabla de Páginas: La tabla de páginas también está representada con un HashMap. Este HashMap tiene como llave el numero de la página y como valor la dirección del marco de página al que apunta. En un inicio todos los valores son -1 ya que no apuntan a ninguna posición en la memoria real.

    - Actualización: La actualización de las tablas de página se lleva a cabo en el actualizador, estas ocurren cuando la matriz (el proceso de suma) necesita que pongan una página en la memoria real. El proceso exacto es el siguiente, cuando la matriz necesita una página, activa una variable en actualizador para que este inicie el proceso. El proceso inicia obteniendo la posición del marco de página a quitar, esto con ayuda de una función que revisa los bytes y el reloj para determinar el que se usó menos reciente y por ende se debería borrar. Una vez tiene ese valor actualiza el marco de página con el nuevo valor y luego actualiza la tabla de páginas de dos formas distintas. Primero a la página que se añadió al marco de página, se le pone como valor el marco de página al que este asociado, y segundo a la página que se borró, se le borra su conexión con el marco de página.

- Memoria Real/Marcos de Página: La memoria real está representada también por un HashMap. Este HashMap tiene como llave un Integer para indicar la posición de la memoria, y como valor otro Integer para indicar la página que está ocupando la posición. En un inicio tienen como valor -1, al no estar ocupada por ninguna página

    - Actualización: La actualización de la memoria real/marcos de página se lleva a cabo en el actualizador, estas ocurren cuando la matriz (el proceso de suma) necesita que pongan una página en la memoria real. El proceso exacto es el siguiente, cuando la matriz necesita una página, activa una variable en actualizador para que este inicie el proceso. El proceso inicia obteniendo la posición del marco de página a quitar, esto con ayuda de una función que revisa los bytes y el reloj

para determinar el que se usó menos reciente y por ende se debería borrar. Una vez tiene ese valor actualiza el marco de página vinculándolo con la nueva página

Como elemento adicional se usó otro HashMap que representa los bits y el reloj para el llamado al algoritmo de envejecimiento. Este HashMap tiene como llave un Integer para representar la posición del marco de página al que hace referencia y long de valor que hace referencia al conteo del reloj y el envejecimiento.

- Actualización: El elemento de envejecimiento se actualiza en la clase *Falla*. Este elemento tiene dos modos de actualización. El primero es el envejecimiento común, el cual ocurre cada milisegundo (o clock tick) y lo que se hace es dividir el número de bits de referencia entre 10 para simular que el bit se corre a la derecha. Ej: 100000 -> 010000. El segundo modo es cuando una nueva página fue añadida o llamada al marco de página, en este modo se le suma 10000000000000, usamos tantos bits para tener más precisión y evitar casos donde el azar sea quien decida cuál es la página más vieja. Además, teniendo en cuenta que en la lectura dice que esperar 160ms es aceptable, nosotros esperamos 16ms por lo que creemos que es suficiente para la prueba.

3. Esquema de sincronización usado. Justifique brevemente dónde es necesario usar sincronización y por qué.

El esquema de sincronización usado es espera activa. Esto ya que los threads están corriendo constantemente revisando si hay algo que actualizar, en el caso del actualizador o envejeciendo en el caso de falla. Además, los threads siempre están revisando por lo que es activo y nunca se ponen a esperar con wait(), solo se ponen a dormir para simular el paso del reloj.

En el proyecto, se usa la sincronización en 1 método.

- *getBytesAsociados():*
  - La sincronización se usa en este método ya que es el único dato que la clase actualizador y falla comparten. Esto ya que falla solo se encarga de envejecer los bytes asociados y actualizador mira los bytes asociados para ver cual página borrar. Entonces ambos no pueden ver los datos al tiempo porque uno los puede estar viendo para modificarlos y el otro para usarlos.

4. Una tabla con los datos recopilados (número de fallas de página por cada caso simulado).

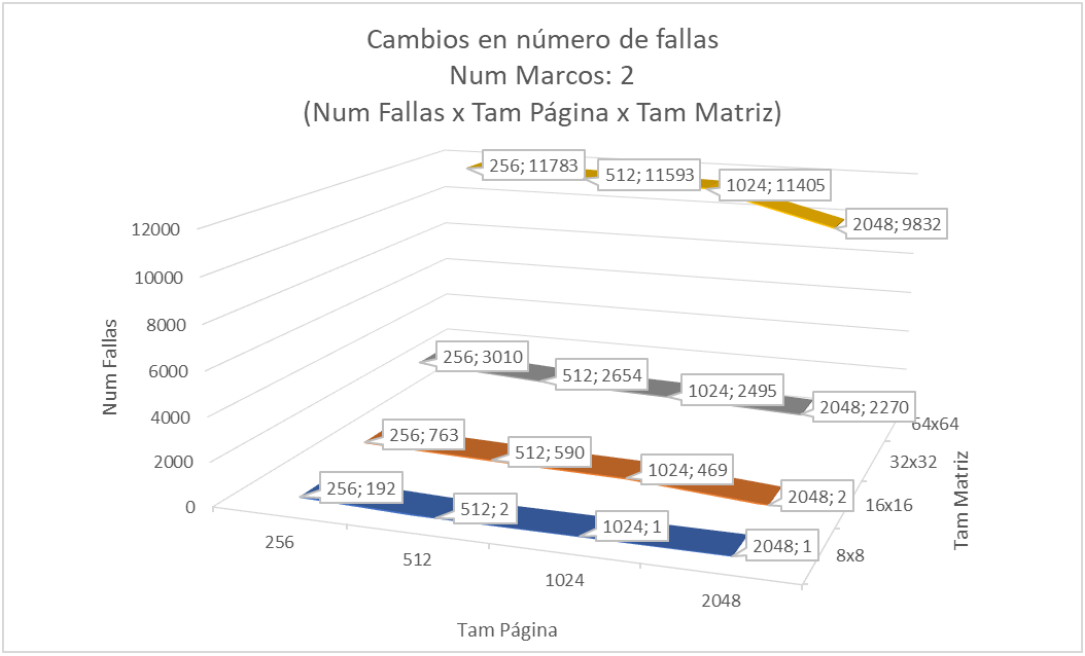
Num Marcos: 2		Tam Matriz			
		8x8	16x16	32x32	64x64
Tam Página	256	192	763	3010	11783
	512	2	590	2654	11593
	1024	1	469	2495	11405
	2048	1	2	2270	9832

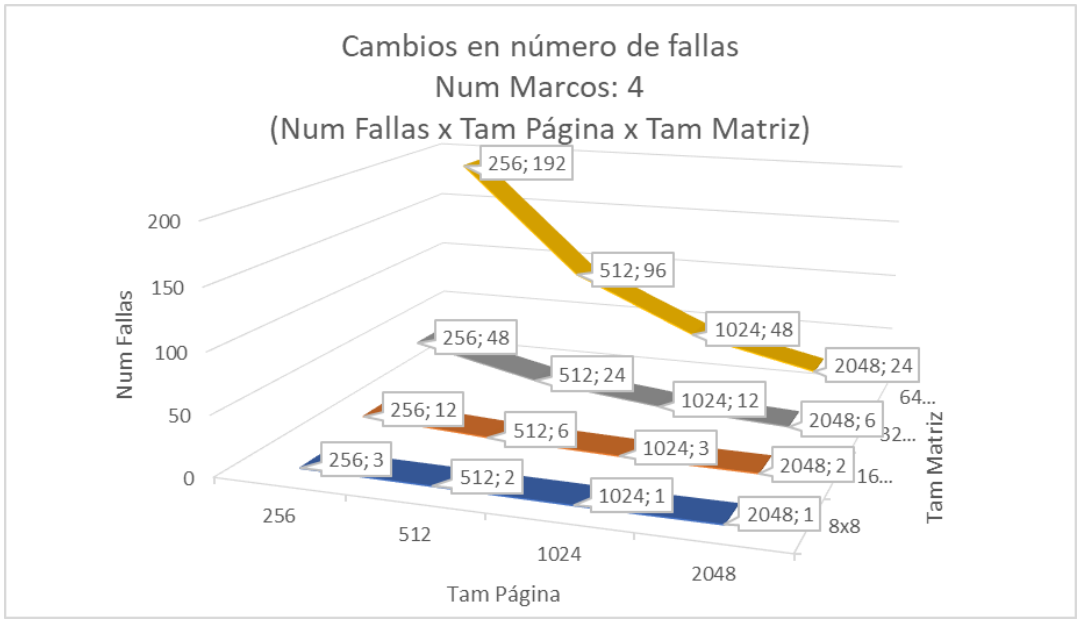
Num Marcos: 4		Tam Matriz			
		8x8	16x16	32x32	64x64
Tam Página	256	3	12	48	192
	512	2	6	24	96
	1024	1	3	12	48
	2048	1	2	6	24

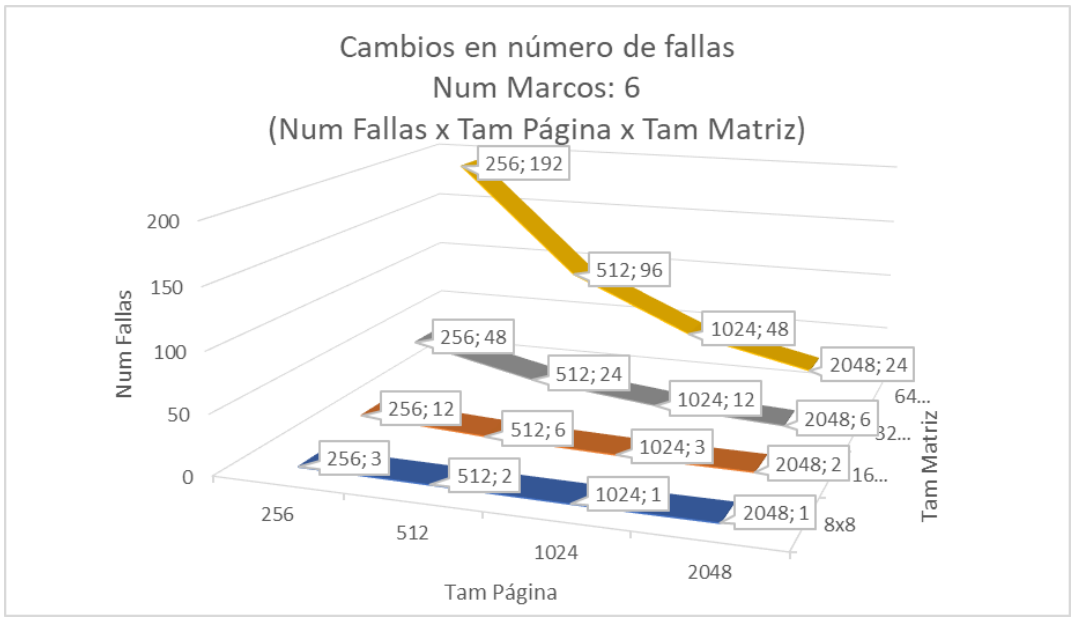
Num Marcos: 6		Tam Matriz			
		8x8	16x16	32x32	64x64
Tam Página	256	3	12	48	192
	512	2	6	24	96
	1024	1	3	12	48
	2048	1	2	6	24

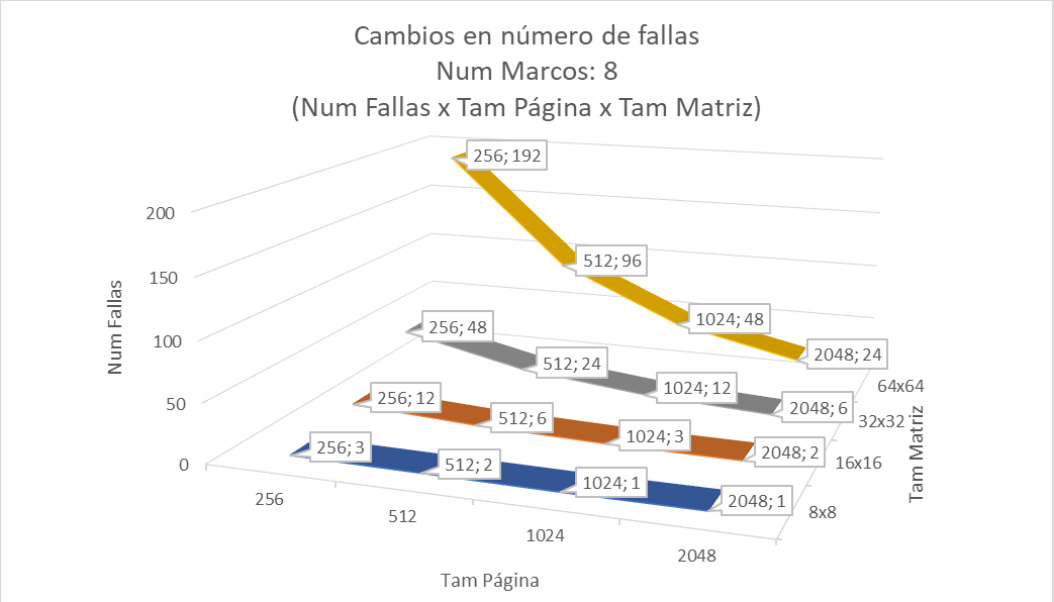
Num Marcos: 8		Tam Matriz			
		8x8	16x16	32x32	64x64
Tam Página	256	3	12	48	192
	512	2	6	24	96
	1024	1	3	12	48
	2048	1	2	6	24

5. **Una serie de gráficas que ilustren el comportamiento del sistema. Para eso cree gráficas en las que fije tamaño de entero y tamaño de página y grafique tamaño de matriz vs. Tamaño de página vs. número de fallas de página.**




Cambios en número de fallas
Num Marcos: 4
(Num Fallas x Tam Página x Tam Matriz)
Num Fallas
200
150
100
50
0
256; 192
512; 96
1024; 48
2048; 24
256; 48
512; 24
1024; 12
2048; 6
256; 12
512; 6
1024; 3
2048; 2
256; 3
512; 2
1024; 1
2048; 1
64...
32...
16...
8x8
Tam Matriz
256
512
1024
2048
Tam Página


Cambios en número de fallas
Num Marcos: 6
(Num Fallas x Tam Página x Tam Matriz)
Num Fallas
200
150
100
50
0
256; 192
512; 96
1024; 48
2048; 24
256; 48
512; 24
1024; 12
2048; 6
256; 12
512; 6
1024; 3
2048; 2
256; 3
512; 2
1024; 1
2048; 1
64...
32...
16...
8x8
Tam Matriz
256
512
1024
2048
Tam Página



6. **Considere otras configuraciones que le permitan entender cómo afecta la memoria virtual el desempeño del programa.**

Una configuración que afecta notablemente el desempeño del programa es tener matrices grandes y tamaños de página pequeños. Lo que esta configuración ocasiona es que se necesiten muchas páginas y que una sola matriz quede almacenada en varias páginas. Esto afecta el desempeño porque constantemente se necesitará cambiar las páginas almacenadas en los marcos de la RAM. Esto se puede observar en todas las gráficas ya que siempre el tamaño más grande de la matriz con el menor número de páginas a evaluar es el que más fallas produce de todas las pruebas que se hizo.

Otro parámetro que podría incidir en el desempeño del programa es el tamaño del entero, ya que, al igual que el caso anterior, si se trabaja con matrices de gran tamaño, el tamaño del entero utilizado puede afectar el consumo de memoria porque un elemento de cada matriz ocuparía más espacio en memoria, potencialmente llevando a más fallos de la página si la memoria es limitada. Esto se puede ver reflejado más detalladamente en la referenciación de páginas, ya que, si se aumenta el tamaño del entero, se necesitarían más páginas ya que cada elemento ocupa una mayor cantidad de bits, por ende, cada página contiene menos elementos.

Por último, la cantidad de marcos de página también puede incidir en el desempeño, ya que si el número de marcos de página es pequeño en comparación con el tamaño de las matrices, puede resultar en una mayor cantidad de fallos, esto sucede porque no habrá suficientes marcos de página para mantener todas las páginas necesarias en memoria. Esto se vio reflejado en las gráficas del punto 5, ya que, por ejemplo, con el menor número de marcos de página y un tamaño muy grande de la matriz, se produjeron fallos muy significativos a comparación de los otros fallos con marcos de página más grandes.

7. **¿Qué pasaría si las matrices fueran almacenadas siguiendo column-major order?**

Si las matrices fueran column-major order, con respecto a la referenciación de las páginas el proceso cambiaría ya que se debe recorrer todas las filas hasta completar una columna para todas las matrices. Sin embargo, con respecto a la determinación de fallas, el proceso no cambiaríañde ninguna manera. Esto ya que la suma de matrices no tiene ningún orden estricto de operaciones, simplemente hay una ley de correspondencia, la cual se seguiría manteniendo. Entonces como todas las matrices son guardadas por columna, simplemente el proceso sumaría

por columnas lo cual al final de todo dejaría el mismo número de errores que los que deja actualmente.

8. ¿Cómo varía el número de fallas de página si el algoritmo estudiado es el de multiplicación de matrices?

Utilizando el algoritmo de envejecimiento para gestionar la paginación de una multiplicación de matrices, es muy probable que este cambio, con respecto a la suma de matrices, aumente el número de los fallos de cada página. La razón principal del porqué esto sucede es porque la multiplicación se utiliza más memoria a comparación de la suma de matrices, ya que implica manipular más datos en la memoria y realizar más cálculos, y esto podría aumentar la posibilidad de que se produzcan fallos en la página. Un ejemplo de cómo se puede ver esto es mediante la referenciación, ya que en todos los casos no tendríamos una matriz C con las mismas dimensiones que la matriz A y B, por lo que dependiendo de la dimensión de la matriz C, se podrían ocupar más espacios en las páginas virtuales.

9. Escriba su interpretación de los resultados: ¿tienen sentido? Justifique sus respuestas

Los resultados sí tienen sentido. En primer lugar, es de esperar que cuando el número de marcos es bajo y el número de páginas es alto, se produzcan más fallos. Esto se comprobó en los casos de prueba, donde al tener 2 marcos de página se presentaron más fallos que cuando se tenían 8 marcos, ambos con la misma cantidad de páginas.