**Caso 2 - Infraestructura Computacional**

Samuel Santa Arias - 202123685
Maria Alejandra Lizarazo - 202021385
Nicolás Díaz Montaña - 202021006

**• Notas antes de empezar:**

- **Para correr el modo uno se debe correr la clase ManejoMemoria.java.**
- **Anexo al programa, hay un zip donde se encuentra el mismo programa pero sin threads. Este archivo se utilizó como base para ver luego como implementarlo en con Threads.**
- **Para correr ese programa se necesita primero configurar en archivo config.txt con la condiciones deseadas. Dos, se corre luego la clase ManejoMemoria.java para crear el output.txt. Tres, por alguna razón se crean dos lineas adicionales al final del archivo output.txt borrelas para que el siguiente paso funcione correctamente. Finalmente, corra la clase ComportamientoSistema.java y este le devolverá el número de fallas. Como nota adicional, si quiere cambiar el marco de página en Comportamiento Sistema diríjase a la variable num_pages.**

**• Descripción del algoritmo Modo 1**

```
public void generarReferenciasPagina() throws IOException {
   Integer tamTotalMatriz = numFilas * numCols * tamEntero;
   Integer numTotalPaginasNecesariasPorMatriz = tamTotalMatriz / tamPagina;

   StringBuilder res2 = new StringBuilder();
   int numeroReferencias = 0;
   for (int i = 0; i < numTotalPaginasNecesariasPorMatriz; i++) {
       int paginaMatA = i;
       int paginaMatB = paginaMatA + numTotalPaginasNecesariasPorMatriz;
       int paginaMatC = paginaMatB + numTotalPaginasNecesariasPorMatriz;
       for (int k = 0; k < numFilas; k++) {
           for (int j = 0; j < numCols; j++) {
               int offSet = (k * numCols + j) * tamEntero % tamPagina;

res2.append("[A-").append(k).append("-").append(j).append("],").append(paginaMatA).
append(",").append(offSet).append("\n");

res2.append("[B-").append(k).append("-").append(j).append("],").append(paginaMatB).
append(",").append(offSet).append("\n");

res2.append("[C-").append(k).append("-").append(j).append("],").append(paginaMatC).
append(",").append(offSet).append("\n");
               numeroReferencias += 3;
           }
       }
   }
```

```
    res1 += "NR=" + numeroReferencias + "\n";
    String respuesta = res1 + res2;
    PrintWriter writer = new PrintWriter(new FileWriter("output.txt", true));
    writer.println(respuesta);
    writer.close();
}
```

Lo que hace el algoritmo es generar los espacios en memoria virtual (paginas) para las matrices A, B y C. Como tal todo el proceso se realiza por medio de la función generarReferenciaPagina(). Lo que se hace primero, es calcular el tamaño total de la matriz utilizando el número de columnas, filas y el tamaño del entero (que generalmente son 4 bytes) que luego se utilizará para saber la cantidad de páginas que se necesitan por matriz. Luego de haber hecho eso, el algoritmo lo que hace es iterar sobre cada página y va generando cada matriz, durante esta iteración se va a calcular el número de página que le corresponde a cada elemento (k, j) de las matrices y su desplazamiento (offSet), para luego almacenarlo en el StringBuilder (del cual se hablará más adelante). Para el desplazamiento se multiplica la posición del elemento en la matriz por el tamaño del elemento y luego se le hace modulo por el tamaño de la página para obtener el desplazamiento en bytes desde el inicio de la página. Finalmente, el algoritmo cuenta el número de multiplicaciones generadas y lo almacena en una variable llamada número de referencia.

**• Descripción de las estructuras de datos usadas**

Para simular el comportamiento del sistema de paginación, utilizamos las siguientes estructuras de datos:

```
Queue<Integer> page_queue = new LinkedList<>();
```

La cola se utiliza para almacenar las referencias de página generadas por el proceso que se está ejecutando. Cada vez que el proceso necesita acceder a una página, se genera una referencia de página y se agrega a la cola. Luego, el sistema de paginación consulta la cola y realiza las operaciones de carga, escritura y reemplazo de página según sea necesario para mantener el conjunto de páginas en memoria real.

La variable "page_queue" es una cola (queue) implementada con una LinkedList en Java. La clase LinkedList es una implementación de la interfaz List en Java, que también puede usarse como una cola a través de su implementación de la interfaz Queue.

```
    List<Integer> references = new ArrayList<>();
```

La variable references es utilizada para almacenar las referencias de página generadas por un proceso. Esta almacena las referencias de página generadas por el proceso que se están utilizando para simular el comportamiento del proceso y del sistema de paginación. Estas referencias se utilizan para tomar decisiones de reemplazo de página basadas en el algoritmo de envejecimiento.

Estas están implementadas mediante el uso de un ArrayList

Además, utilizamos la clase StringBuilder de java, para construir y manipular cadenas de caracteres de manera más eficiente, ya que se demoraba mucho si se hacía una concatenación de los strings. Principalmente debido a que se tiene ciclos for anidados, lo cual hace que la complejidad aumente hasta O(n^2).

**• Esquema de sincronización usado.**
En la implementación propuesta se usó la sincronización y los threads en la clase Aging para garantizar que la actualización de las edades de las páginas y el envejecimiento se realizarán en un hilo separado y no interfirieran con la simulación de la política de reemplazo de página que se realiza en la clase ComportamientoSistema.

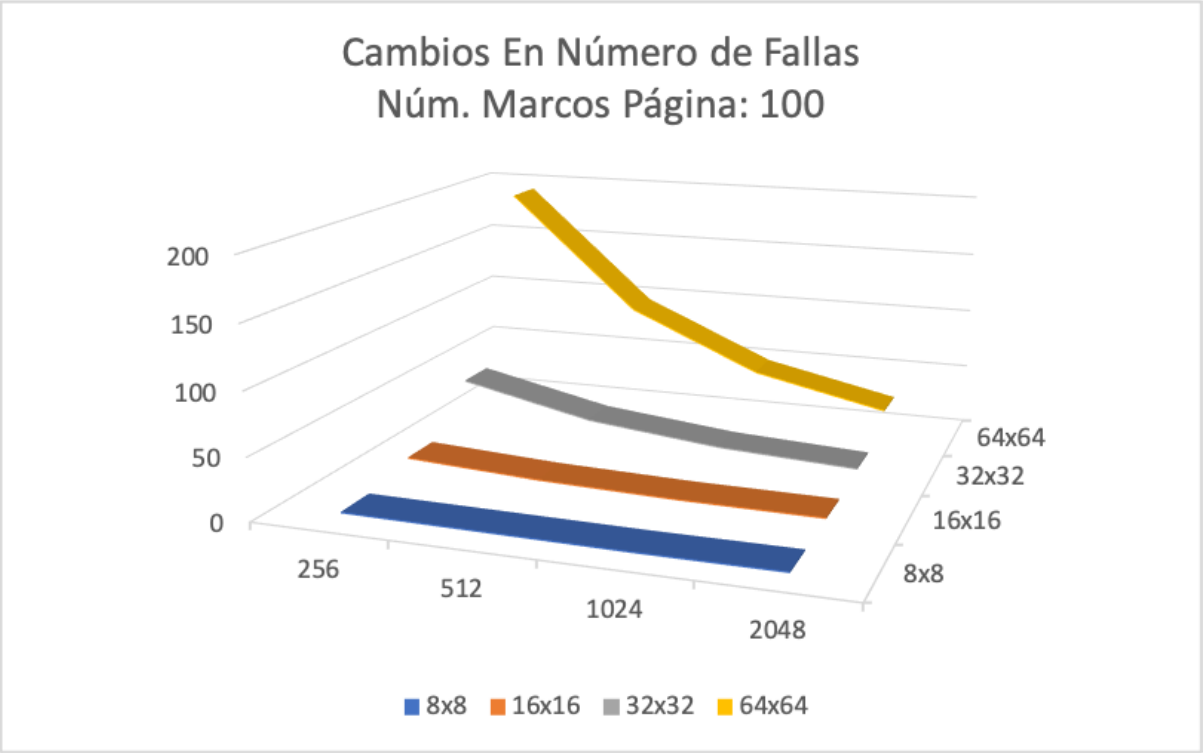
La sincronización se utilizó en el método updateAges de la clase Aging para garantizar que la actualización de las edades de las páginas se realizará de forma segura y no se produjeran condiciones de carrera. Además, se utilizó el mecanismo de espera-notify para que el hilo de envejecimiento espere a que se actualicen las edades de las páginas antes de realizar el envejecimiento.

Se utilizaron threads para permitir que la actualización de las edades y el envejecimiento se realizarán en paralelo con la simulación de la política de reemplazo de página en la clase ComportamientoSistema, lo que permite una mayor eficiencia y uso del procesador. Además, la separación de las tareas en hilos separados mejora la modularidad y legibilidad del código.

**• Tabla con los datos recopilados y gráficos que demuestran el comportamiento del sistema.**

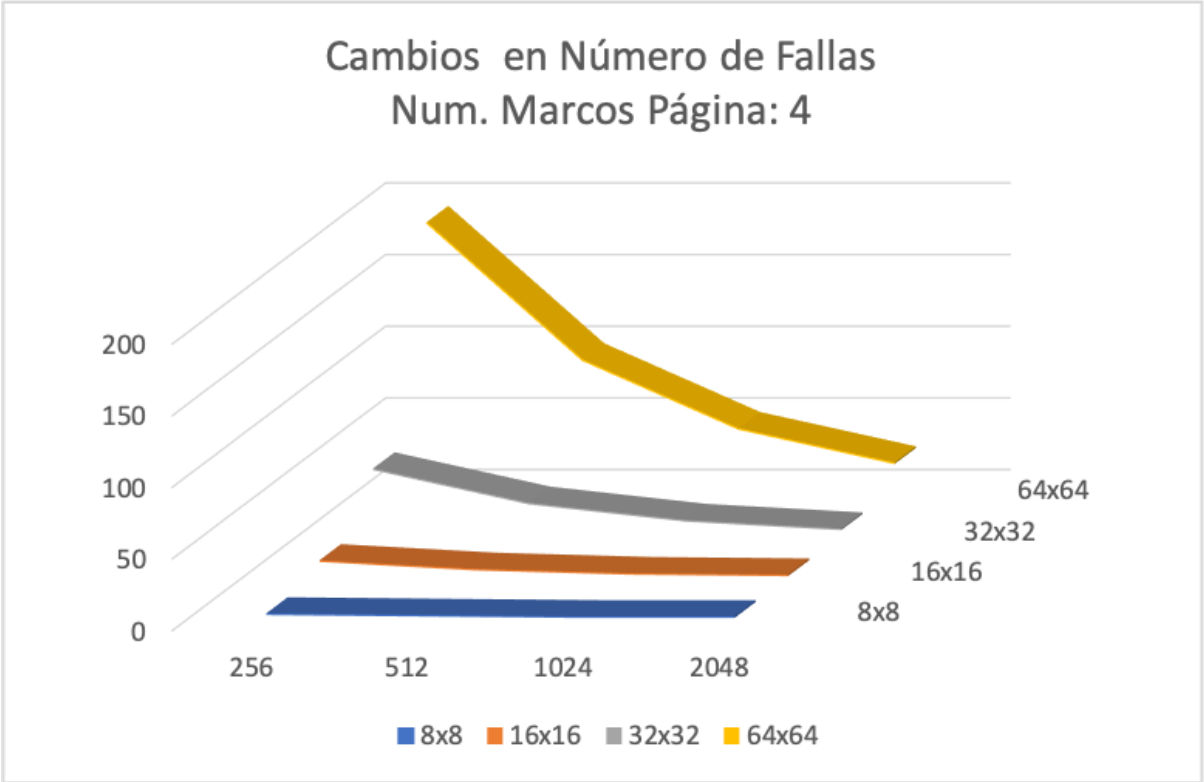
Marco de Página 100

| | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|
| 8x8 | 3 | 2 | 1 | 1 |
| 16x16 | 12 | 6 | 3 | 2 |
| 32x32 | 48 | 24 | 12 | 6 |
| 64x64 | 192 | 96 | 48 | 24 |



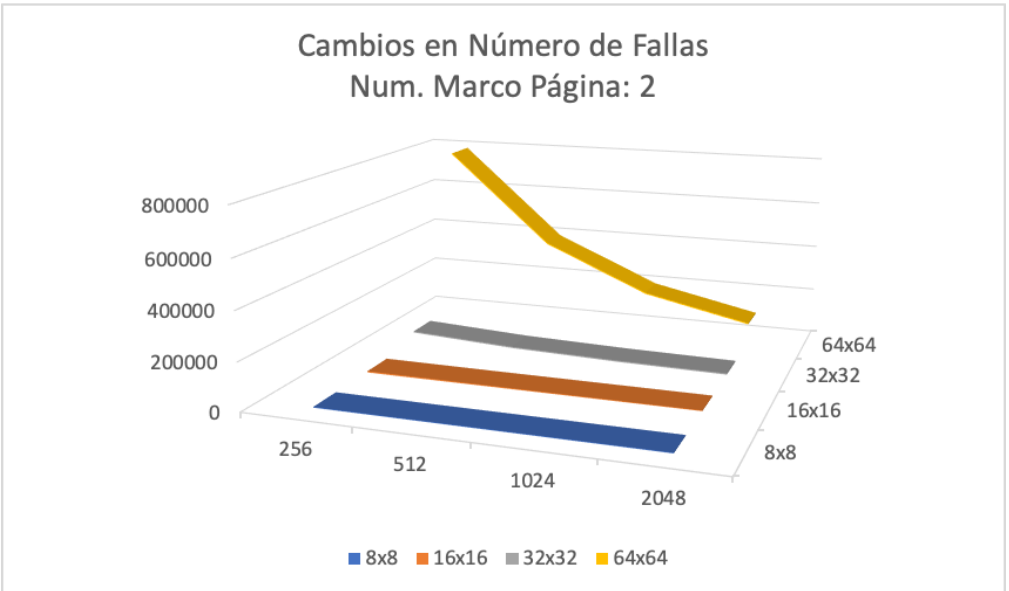

Marco de Página 4

| | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|
| 8x8 | 3 | 2 | 1 | 1 |
| 16x16 | 12 | 6 | 3 | 2 |
| 32x32 | 48 | 24 | 12 | 6 |
| 64x64 | 192 | 96 | 48 | 24 |



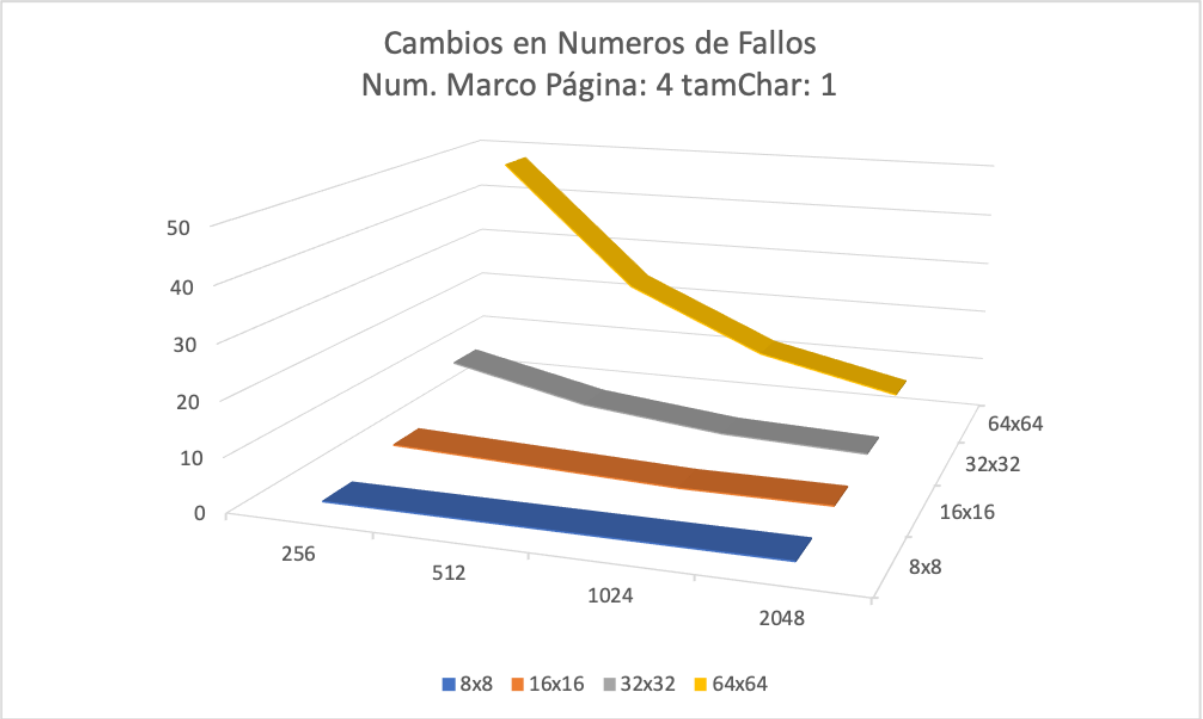

Marco de Página 2

| | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|
| 8x8 | 192 | 2 | 1 | 1 |
| 16x16 | 3072 | 1536 | 768 | 2 |
| 32x32 | 49152 | 24576 | 12288 | 6144 |
| 64x64 | 786432 | 393216 | 196608 | 98304 |



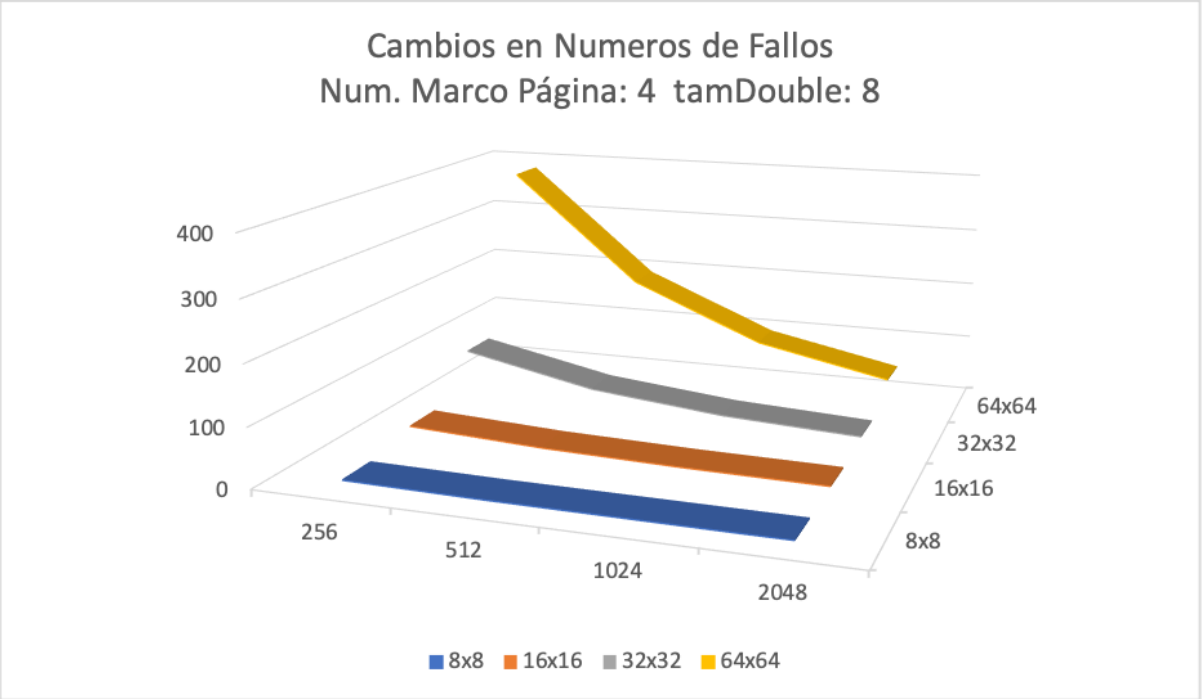
Marco de Página 4, tamaño de un char (1byte)

| | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|
| 8x8 | 1 | 1 | 1 | 1 |
| 16x16 | 3 | 2 | 1 | 1 |
| 32x32 | 12 | 6 | 3 | 2 |
| 64x64 | 48 | 24 | 12 | 6 |




Marco de Página 4, tamaño de un double (8 bytes)

| | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|
| 8x8 | 6 | 3 | 2 | 1 |
| 16x16 | 24 | 12 | 6 | 3 |
| 32x32 | 96 | 48 | 24 | 12 |
| 64x64 | 382 | 192 | 96 | 48 |




**• ¿Qué pasaría si las matrices fueran almacenadas siguiendo column-major order?**
Si las matrices fueran almacenadas siguiendo column-majo, significaría que los elementos de la matriz se almacenan en columnas consecutivas en lugar de filas consecutivas. En este caso, el acceso a los elementos de la matriz sería diferente, ya que el acceso se realizaría en columnas consecutivas.

Para el programa de simulación de paginación, esto podría afectar el rendimiento de la simulación de paginación, ya que el algoritmo de envejecimiento se basa en la frecuencia de acceso a las páginas, y si los elementos de la matriz están en columnas consecutivas, la frecuencia de acceso a las páginas podría ser diferente.

**• ¿Cómo varía el número de fallas de página si el algoritmo estudiado es el de multiplicación de matrices?**

El número de fallas podría variar dependiendo del algoritmo usado, en especial si a las matrices se accede de manera secuencial, (el cual podría ayudar a reducir el número de fallas) o de manera recursiva, donde se podría aumentar el número de fallas de páginas.

**• Escriba su interpretación de los resultados: ¿tienen sentido? Justifique su respuesta.**

En los resultados se puede evidenciar cómo el algoritmo organiza de manera eficiente el espacio de la memoria real. En este caso, se puede ver que entre más espacio tenga las páginas menos fallas va a ver. Esto tiene sentido dado a que la memoria virtual tiene lo que necesita para guardar las página de memoria virtual. A si mismo es su contrario, se puede ver que entre menos espacio tenga la página real más fallos se va a producir. Finalmente también los resultados se ve afectado por el tamaño del entero, que este caso utilizamos el tamaño de un double y el de un char. Donde se evidencia que se dependiendo del espacio

que necesite una matriz se va necesitar más paginas y por ende puede producir más fallos de página.