**Caso 1**

**William Bayona 202011494**

**Amelia Serrano 202221556**

**23 de septiembre**

**2025-2**

**Tecnologia de Infraestructura y Computo**

**Seccion 3**

1. **Descripción del algoritmo usado para generar las referencias de pagina (opción uno)**

El algoritmo implementado para generar las referencias de página inicia leyendo un archivo de configuración que define el tamaño de página, el número de procesos y las dimensiones de las matrices que cada proceso manipulará. A partir de esos datos, el programa asigna a cada proceso un espacio virtual contiguo donde se ubican las tres matrices en orden: primero matriz1, luego matriz2 y finalmente matriz3. Este almacenamiento se hace en formato row-major order (por filas), lo que facilita calcular las posiciones de cada elemento. Luego, el algoritmo itera sobre cada elemento (i, j) de las matrices y genera tres accesos distintos: lectura de matriz1[i][j], lectura de matriz2[i][j] y escritura en matriz3[i][j]. Para calcular la dirección virtual de cada acceso se tiene en cuenta que cada entero ocupa 4 bytes, por lo tanto, la dirección base se multiplica por el desplazamiento correspondiente, aplicando un desfase adicional según si se trata de la segunda o tercera matriz. El resultado es un conjunto secuencial de direcciones que se traducen en páginas virtuales, desplazamientos y tipo de operación.

En la implementación se busca reforzar esta lógica con bucles que calculan de manera explícita las direcciones virtuales de cada acceso, sumando los desplazamientos correspondientes cuando se trata de la segunda y tercera matriz. El resultado se escribe en un archivo proc<i>.txt por proceso, incluyendo primero una cabecera con los metadatos (TP, NF, NC, NR, NP) y luego la lista completa de referencias en el formato definido: matriz y celda, página virtual, desplazamiento y tipo de operación (r/w). De esta forma, no solo se simula el comportamiento real de los accesos a memoria de un programa que suma matrices, sino que además se preparan entradas detalladas para la siguiente etapa de la simulación de memoria virtual, que usará esas referencias para evaluar fallos de página, hits y operaciones de swap.

1. **Descripción de las estructuras de datos usadas para simular el comportamiento del sistema de paginación**

Para simular el comportamiento de un sistema de paginación por demanda, el programa se apoya en un conjunto de clases que modelan los componentes principales de un sistema operativo real, es decir, procesos, tablas de páginas y el orquestador de la simulación. La clase Proceso encapsula todo el contexto de memoria de un programa, incluyendo tres estructuras claves. Primero, el Map<Integer, EntradaTablaPaginas> tablaPaginas, que funciona como la tabla de páginas del proceso, mapeando directamente los números de página virtual a su estado en memoria. Esta estructura se consulta en cada referencia de memoria para decidir si la página ya está cargada (hit) o si debe producirse un fallo de página. Su actualización ocurre solo en caso de fallos: se marca la entrada como válida y se le asigna un marco físico, mientras que la página víctima, en caso de reemplazo, se invalida. Además, el proceso mantiene un ArrayList<String> referencias que actúa como la lista ordenada de accesos de memoria a procesar, cargada al inicio desde el archivo proc<i>.txt y recorrida durante toda la simulación. Finalmente, gestiona un List<Integer> marcosAsignados, que representa los marcos físicos permitidos para ese proceso y que puede crecer si otros procesos terminan y liberan memoria.

El detalle de cada página virtual está representado por la clase EntradaTablaPaginas. Cada entrada contiene atributos básicos como el número de marco asignado y un bit de validez, ademas del long contadorLRU. Este contador implementa el algoritmo de reemplazo por Least Recently Used. En cada acceso a memoria, los contadores de todas las páginas válidas se desplazan un bit a la derecha, simulando el paso del tiempo, y a la página accedida se le enciende el bit más significativo. Así, las páginas usadas recientemente mantienen valores altos, mientras que las menos usadas envejecen y terminan siendo candidatas naturales a reemplazo. Así, cuando ocurre un fallo de página y no hay marcos libres, el simulador puede seleccionar la víctima con el menor valor de contadorLRU.

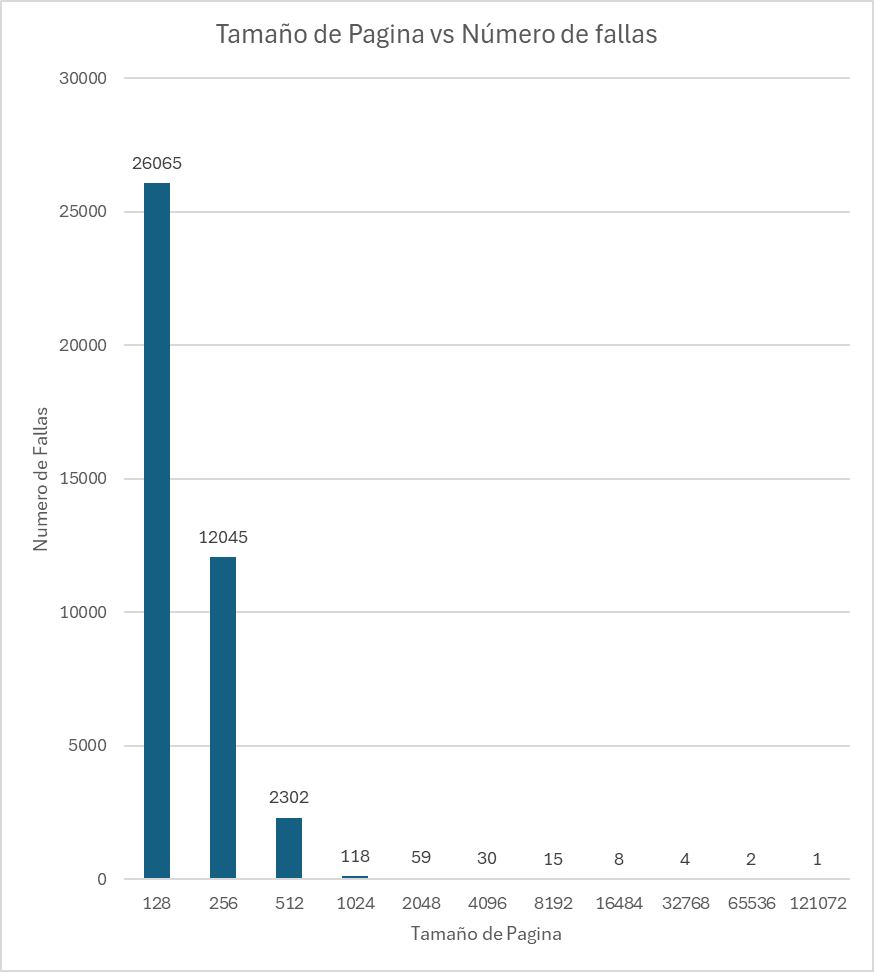
Por último, la clase Simulador coordina el funcionamiento global. Su pieza central es la cola de procesos (Queue<Proceso> colaProcesos). En cada ciclo, el simulador extrae un proceso de la cabeza de la cola, procesa una referencia de memoria y, dependiendo del resultado, lo vuelve a insertar al final si aún le quedan accesos pendientes. Cuando un proceso termina, libera sus marcos de página, que se reasignan dinámicamente al proceso en ejecución con mayor número de fallos, maximizando la eficiencia de la memoria. De esta forma, el simulador integra los distintos elementos: las tablas de páginas permiten decidir hits y fallos, las estructuras LRU controlan la sustitución y la cola asegura la planificación justa de los procesos

1. **Pruebas con matrices 100x100 variando el tamaño de pagina.**

Se realizaron las respectivas pruebas utilizando únicamente matrices 100x100 teniendo en cuenta los siguientes supuestos como parámetros del programa.

* Tenemos un único proceso
* Dicho proceso tiene una matriz 100x100
* Partiremos las pruebas con un tamaño de pagina de 128 y se ira duplicando (256,512,1024,2048…)
* Le asignaremos un total de 128 marcos de página en la RAM

Ya con estos supuestos se realizaron las pruebas y se obtuvo el siguiente resultado.



Tamaño de Pagina vs Número de Fallas

En la gráfica se evidencia que al aumentar el número de procesos con una memoria RAM fija, el rendimiento se degrada rápidamente. Cada proceso recibe menos memoria, lo que le impide mantener sus datos y genera una alta competencia por el espacio.

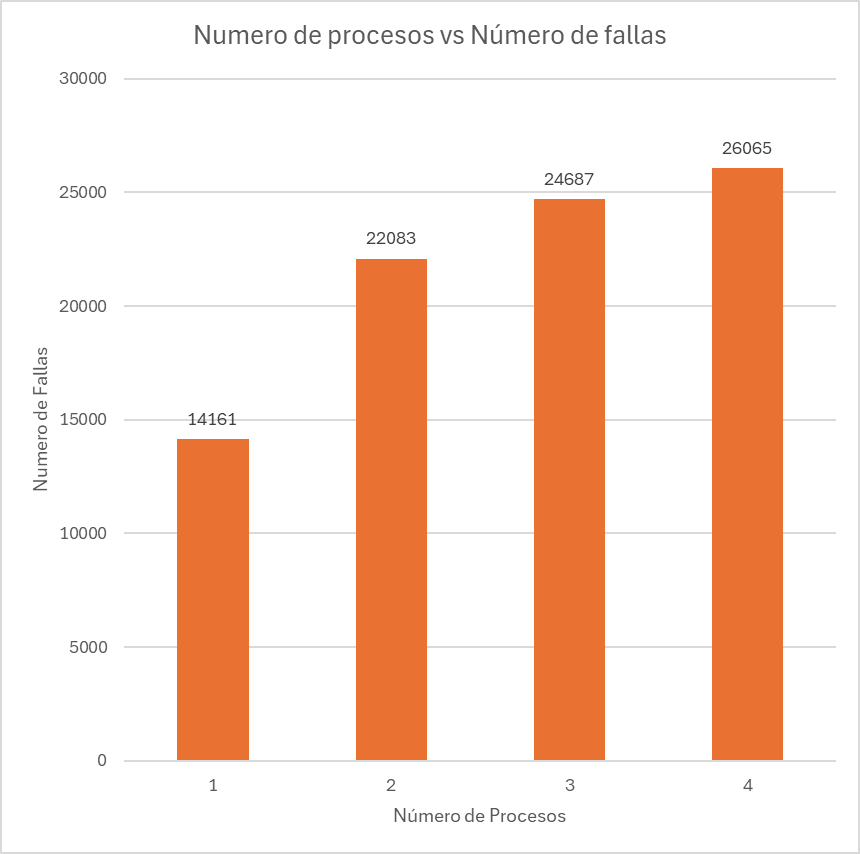
1. **Cuatro Nuevas Configuraciones**
2. **Variando Numero de procesos:**

Variaremos el número de procesos (1,2,3,4)

Todos los procesos tienen matrices 100x100

Le asignaremos un total de 512 marcos

Tendremos un tamaño de página de 128



Numero de procesos vs Numero de fallas

En la gráfica se evidencia que al aumentar el número de procesos con una memoria RAM fija, el rendimiento se degrada igualmente. Cada proceso recibe menos memoria, lo que le impide mantener sus datos esenciales y genera una alta competencia por el espacio. Esto fuerza a los procesos a un ciclo ineficiente de intercambio de datos con el disco, provocando que el sistema dedique más tiempo a gestionar la memoria que a ejecutar tareas.

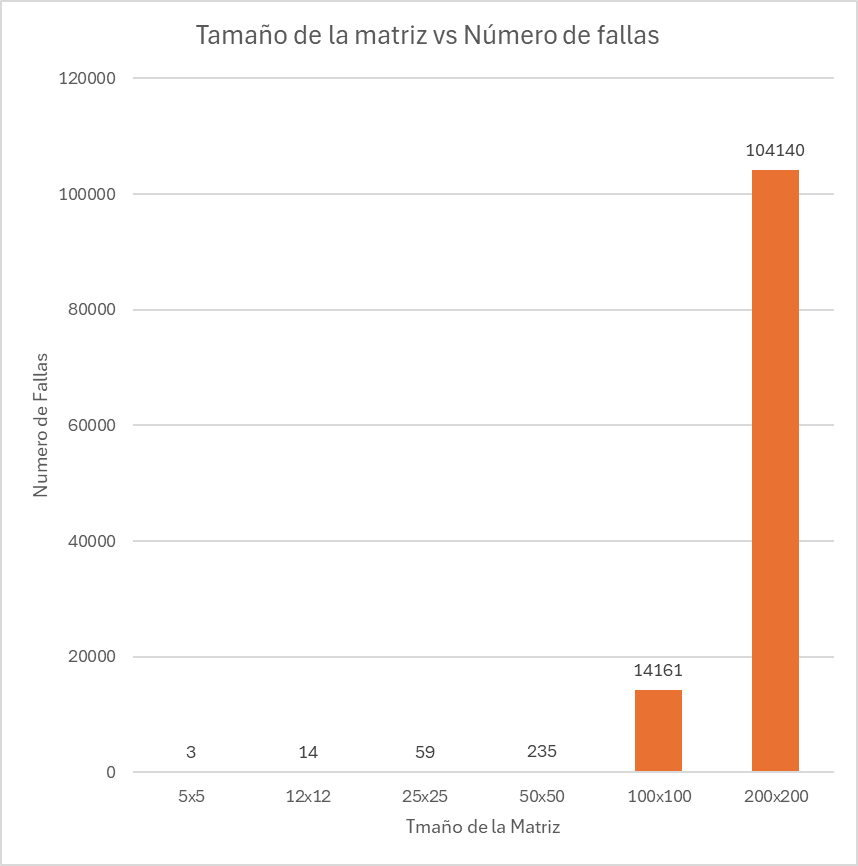
1. **Variando el tamaño de la Matriz:**

Tenemos un único proceso

Variaremos el tamaño de la matriz (5x5, 12x12, 25x25,100x100,200x200)

Le asignaremos un total de 512 marcos

Tendremos un tamaño de página de 128



Tamaños de la matriz vs Numero de fallas

La gráfica muestra que a mayor tamaño de la matriz, mayor es el número de fallos de página.

Es el resultado esperado, ya que una matriz más grande ocupa un mayor espacio en la memoria virtual y, por lo tanto, requiere un número superior de páginas únicas para almacenar todos sus datos. Con una cantidad fija de memoria RAM, un programa más grande necesita cargar y descargar páginas con más frecuencia, ya que su "working set" (conjunto de páginas activas) es más extenso. Cada página única que el programa necesita debe cargarse al menos una vez, lo que incrementa inevitablemente el conteo de "fallos obligatorios" y la probabilidad de fallos por reemplazo.

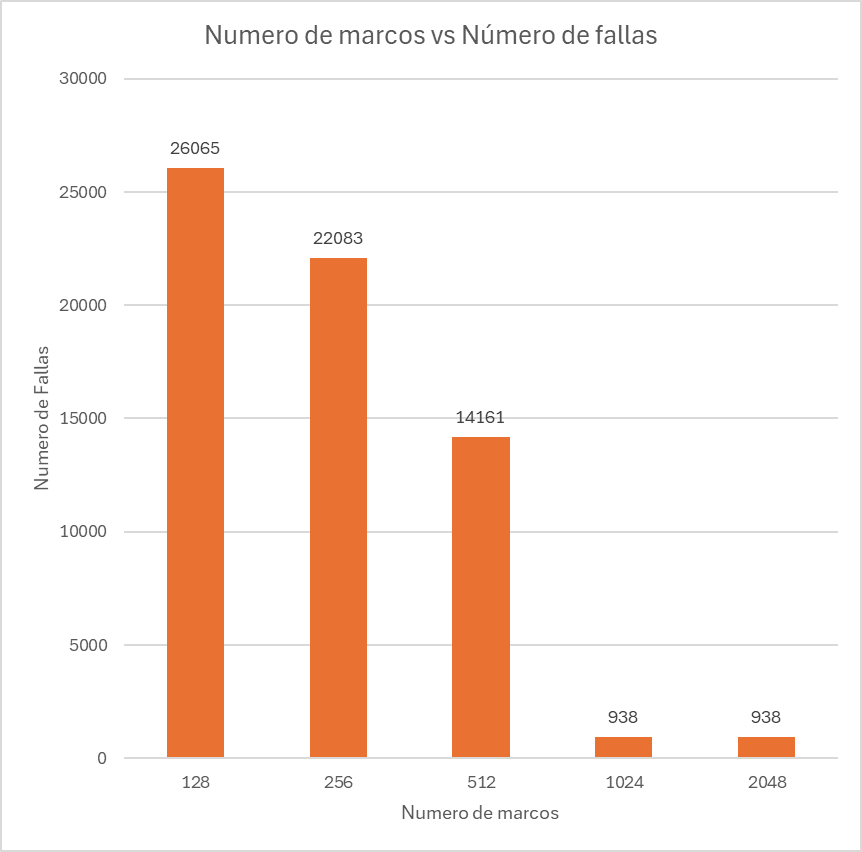
1. **Variando el número de marcos:**

Tenemos un único proceso

Todos los procesos tienen matrices 100x100

Variaremos el número de marcos (128, 256,512,1024,2048…)

Tendremos un tamaño de página de 128



El análisis de la gráfica confirma la relación inversa entre la cantidad de memoria física asignada (marcos de página) y la incidencia de fallos. Con un número reducido de marcos, el sistema no puede contener el conjunto de trabajo del proceso, lo que exige un acceso constante al disco y resulta en una alta tasa de fallos. A medida que se incrementa el número de marcos, esta tasa disminuye rápidamente.

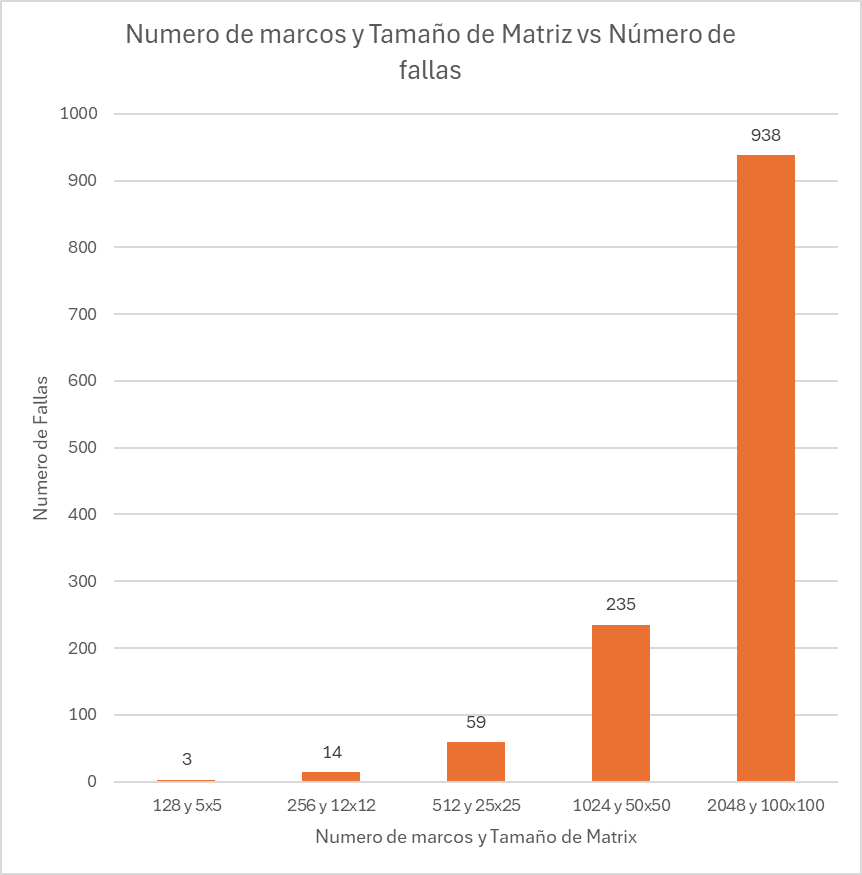
1. **Variando el tamaño de la matriz y el número de marcos:**

Tenemos un único proceso

Variaremos el tamaño de la matriz (100x100, 200x200, 300x300,400x400)

Variaremos el número de marcos (128, 256,512,1024,2048…)

Tendremos un tamaño de página de 128



La gráfica muestra dos conclusiones claras: primero, para una cantidad de memoria fija, **una matriz más grande siempre genera más fallos**, ya que su mayor tamaño requiere más páginas únicas. Segundo, independientemente del tamaño de la matriz, **aumentar la memoria RAM siempre reduce los fallos**.

El peor escenario es el de una matriz grande con poca memoria (200x200 con 64 marcos), donde la alta demanda de memoria supera con creces los recursos, causando un número masivo de fallos. Esto demuestra que la eficiencia no depende de un solo factor, sino de la proporción entre la memoria que un proceso necesita y la que se le asigna.

1. **Pregunta: ¿Cómo se afecta el tiempo de respuesta de un proceso por el numero de accesos a SWAP?**

El tiempo de respuesta de un proceso si se ve afectado por la cantidad de accesos a SWAP, ya que cada vez que ocurre un fallo de página el sistema debe traer la página faltante desde memoria secundaria al marco de RAM asignado. Este proceso es mucho más lento que un acceso normal a memoria física, pues implica operaciones de lectura y, en caso de reemplazo, también de escritura en disco. Por lo tanto, a mayor número de accesos a SWAP, el tiempo total de ejecución del proceso aumenta, retrasando la finalización de sus instrucciones y reduciendo la eficiencia del sistema en general.

1. **Pregunta: ¿Sumar matrices representa un problema de localidad alta, meda o baja? (Parrafo explicando no mas de 10 lineas)**

Sumar matrices es un problema de localidad alta, porque el acceso a memoria se hace de forma muy ordenada y repetitiva. Es decir, el programa recorre las matrices por filas, así que va usando elementos que están pegados en memoria uno tras otro. Esto aprovecha la localidad espacial, ya que cuando se carga una página, casi todos los datos que contiene se van a usar de inmediato. También hay un poco de localidad temporal, porque para cada posición (i, j) primero se leen los valores de las dos matrices originales y enseguida se guarda el resultado en la tercera. En otras palabras, el patrón de acceso es bastante predecible y eficiente, lo que hace que haya menos fallos de página.