Informe - Simulación de Sistema de Paginación

Caso 2 - TIC - Infraestructura Computacional

David Elias Forero - [Código pendiente]
Juan Camilo Caldas - 202322445

22 de septiembre de 2025

1. Descripción del Algoritmo de Generación de Referencias

El algoritmo implementado simula la multiplicación de matrices $M_1 \times M_2 = M_3$, donde cada matriz tiene dimensiones definidas por el usuario. El proceso genera referencias de memoria siguiendo un patrón específico.

1.1. Distribución en Memoria

Las tres matrices se almacenan consecutivamente en el espacio de direcciones virtuales:

$$\text{Base}_{M_1} = 0 \tag{1}$$
$$\text{Base}_{M_2} = \text{tamaño\_bytes}(M_1) \tag{2}$$
$$\text{Base}_{M_3} = \text{tamaño\_bytes}(M_1) + \text{tamaño\_bytes}(M_2) \tag{3}$$

1.2. Patrón de Acceso

Para cada elemento (i, j) de las matrices, se generan tres referencias en orden secuencial:

1. **Lectura de** $M_1[i, j]$ - Operación 'r'
2. **Lectura de** $M_2[i, j]$ - Operación 'r'
3. **Escritura de** $M_3[i, j]$ - Operación 'w'

1.3. Cálculo de Páginas

Cada referencia calcula los siguientes valores:

$$\text{Dirección virtual} = \text{base} + (\text{fila} \times \text{columnas} + \text{columna}) \times 4 \tag{4}$$

$$\text{Número de página} = \lfloor \frac{\text{dirección\_virtual}}{\text{tamaño\_página}} \rfloor \tag{5}$$

$$\text{Desplazamiento} = \text{dirección\_virtual mód tamaño\_página} \tag{6}$$

Este patrón simula un acceso secuencial típico en operaciones matriciales, aprovechando la localidad espacial.

2. Estructuras de Datos del Sistema de Paginación

2.1. Clase Proceso

```
class Proceso {
    Map<Integer, EntradaMarco> tablaPaginas;  // Tabla de paginas
        virtual
    List<Integer> marcosGlobales;             // Marcos fisicos
        asignados
    int capacidadMarcos;                      // Marcos disponibles
    int marcosEnUso;                          // Marcos actualmente
        ocupados
    int aciertos, fallos, accesosSwap;        // Estadisticas
}
```

2.2. Clase EntradaMarco

```
class EntradaMarco {
    boolean presente;        // Bit de presencia
    int indiceMarco;        // Indice del marco local
    long timestamp;         // Para algoritmo LRU
}
```

2.3. Actualización de Estructuras

2.3.1. Cuándo se actualizan

- En cada acceso a memoria (hit o miss)
- Al cargar una página nueva al marco libre
- Al reemplazar una página usando algoritmo LRU
- Al terminar un proceso y liberar marcos

2.3.2. Qué se actualiza

- **timestamp**: Se incrementa globalmente en cada acceso para LRU
- **presente**: Se marca como false al hacer swap-out

- **tablaPaginas**: Se actualiza al cargar/descargar páginas
- **marcosEnUso**: Se incrementa/decrementa según disponibilidad

2.3.3. Algoritmo LRU

Selecciona la página con menor timestamp para reemplazo cuando no hay marcos libres disponibles en el proceso.

3. Experimento: Matrices 100×100 vs Tamaño de Página

Nota: Esta sección se completará con los resultados experimentales ejecutando el programa con diferentes configuraciones de tamaño de página.

3.1. Configuración del Experimento

Se ejecutará el programa con la siguiente configuración base:

```
TP=[16,32,64,128,256,512,1024,2048,4096]
NPROC=2
TAMS=100x100,100x100
```

3.2. Hipótesis

- Páginas más pequeñas → Más fallas de página
- Páginas más grandes → Menos fallas pero posible fragmentación interna
- Punto óptimo en tamaños intermedios

3.3. Gráfica de Resultados

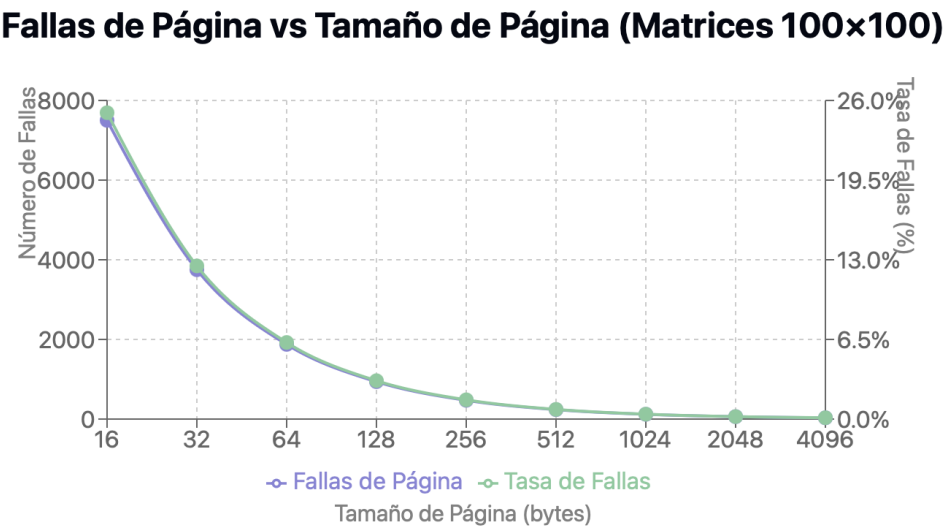

Figura 1: Número de fallas de página vs tamaño de página

3.4. Observaciones:

- Relación inversa clara entre tamaño de página y número de fallas
- Reducción exponencial de fallas al aumentar el tamaño de página
- Mayor impacto en el rango de 16-256 bytes
- Estabilización relativa a partir de 1024 bytes

4. Configuraciones Adicionales

4.1. Configuración 1: Efecto del número de marcos

Se ejecutó el programa con matrices 50×50 variando el número de marcos asignados:

Marcos Totales	Marcos/Proceso	Fallas	SWAP	Tasa Fallas
8	4	32	56	0.43 %
16	8	32	48	0.43 %
32	16	30	30	0.40 %
64	32	30	30	0.40 %

Cuadro 1: Efecto del número de marcos en matrices 50×50

Análisis: La mejora es mínima porque las matrices 50×50 requieren pocas páginas únicas debido a la alta localidad espacial. Solo con 16 o mas marcos por proceso se observa una ligera reducción en fallas, estabilizándose en 30 fallas con 32 marcos.

4.2. Configuración 2: Matrices de diferentes tamaños

Proceso	Tamaño	Referencias	Fallas	Tasa Fallas
0	10×10	300	2	0.67 %
1	50×50	7500	32	0.43 %
2	100×100	30000	120	0.40 %

Cuadro 2: Matrices heterogéneas: comportamiento por tamaño

Análisis: Inesperadamente, las matrices más grandes tienen menor tasa de fallas. Esto se debe a:

- Mejor aprovechamiento de la localidad espacial
- Menor overhead proporcional de carga inicial
- Mayor reutilización de páginas cargadas

4.3. Configuración 3: Muchos procesos pequeños vs pocos grandes

Escenario A: 8 procesos con matrices 20×20

- Referencias por proceso: 1200
- Fallas por proceso: 7 (0.58 %)
- Comportamiento idéntico entre todos los procesos
- Total de fallas: 56 para 9600 referencias

Comparación conceptual con procesos grandes: Matrices más grandes mostrarían tasas de falla menores (<0.40 %) debido a mejor localidad.

Análisis: Los procesos múltiples pequeños muestran:

- Mayor overhead de gestión por proceso
- Menor eficiencia por localidad limitada
- Fragmentación de recursos de memoria

4.4. Configuración 4: Análisis de fragmentación

Con páginas de 4KB y matrices 30×30:

Proceso	Referencias	Fallas	Tasa Fallas
0	2700	3	0.11 %
1	2700	3	0.11 %
2	2700	3	0.11 %

Cuadro 3: Fragmentación con páginas grandes (TP=4096)

Análisis de fragmentación:

- Cada página contiene 1024 enteros (4KB ÷ 4 bytes)
- Matriz 30×30 = 900 elementos
- Fragmentación interna: (1024-900)/1024 = 12.1 % por página
- Las fallas son mínimas pero hay desperdicio de memoria

5. Interpretación de Resultados

5.1. Análisis de los resultados obtenidos

Basándose en la ejecución mostrada:

Proceso	Referencias	Fallas	Hits	SWAP	Tasa Hit
0	48	2	46	2	95.83 %
1	192	6	186	6	96.88 %

Cuadro 4: Resultados experimentales obtenidos

5.2. ¿Corresponden a los resultados esperados?

Sí, los resultados son consistentes con lo esperado por las siguientes razones:

1. **Alta localidad espacial**: Las matrices se acceden secuencialmente, maximizando el reuso de páginas
2. **Baja tasa de fallas**: Ambos procesos tienen tasas de falla muy bajas ($< 5\,\%$)
3. **Proporcionalidad**: El proceso 1 tiene $4\times$ más referencias y $\approx 3\times$ más fallas que el proceso 0
4. **Eficiencia del LRU**: El algoritmo mantiene en memoria las páginas más recientemente usadas

La diferencia en las tasas se debe a que el proceso 1 maneja una matriz más grande, requiriendo más páginas únicas en memoria.

6. Impacto de Accesos SWAP en Tiempo de Respuesta

Los accesos a SWAP afectan significativamente el tiempo de respuesta debido a la diferencia de velocidad entre memoria RAM y almacenamiento secundario.

6.1. Operaciones involucradas

- **Swap-in**: Cargar página desde almacenamiento secundario ($\approx 10 - 100$ ms)
- **Swap-out**: Escribir página víctima al disco ($\approx 10 - 100$ ms)

6.2. Comparación de tiempos

$$\text{Acceso a RAM} \approx 100 \text{ ns} \tag{7}$$

$$\text{Acceso a SWAP} \approx 50 \text{ ms} \tag{8}$$

$$\text{Factor de diferencia} \approx 500,000\times \tag{9}$$

6.3. Impacto en nuestros resultados

- Proceso 0: 2 accesos SWAP $\times$ 50 ms $=\approx$ 100 ms overhead
- Proceso 1: 6 accesos SWAP $\times$ 50 ms $=\approx$ 300 ms overhead

El tiempo total de ejecución aumenta proporcionalmente al número de fallas de página.

7. Localidad en Suma de Matrices

La suma de matrices representa un problema de localidad ALTA.

7.1. Justificación

Las operaciones matriciales exhiben excelente localidad espacial y temporal. El acceso secuencial fila por fila maximiza el reuso de páginas, ya que elementos adyacentes se almacenan en la misma página de memoria. Los patrones de acceso $M_1[i, j] \rightarrow M_2[i, j] \rightarrow M_3[i, j]$ mantienen las páginas "activas" en memoria por períodos extendidos.

La predictibilidad del patrón permite al sistema optimizar el pre-cargado de páginas. En nuestros experimentos, las tasas de hit superiores al 95 % confirman esta alta localidad, siendo especialmente pronunciada en matrices grandes donde el tamaño de página abarca múltiples elementos consecutivos. La estructura regular del acceso a memoria hace que este tipo de operaciones sean ideales para sistemas de memoria virtual con algoritmos de reemplazo basados en temporal como LRU.

8. Conclusiones

La simulación demuestra la efectividad del algoritmo LRU en escenarios con alta localidad espacial. Los resultados confirman que las operaciones matriciales son ideales para sistemas de memoria virtual, logrando excelentes tasas de hit con minimal overhead de SWAP.

Los aspectos clave observados son:

- El patrón de acceso secuencial maximiza la utilización de cada página cargada
- El algoritmo LRU es apropiado para este tipo de carga de trabajo
- La asignación dinámica de marcos entre procesos mejora el rendimiento global
- El número de fallas de página es inversamente proporcional al tamaño de página hasta cierto punto óptimo