Análisis documental

Caso 2 — Simulador de Memoria Virtual

Zair Samuel Montoya Bello (código: 202321067)
Tomás Hernández (código: 202320326)

24 de septiembre de 2025

Introducción

En este apartado se explica cómo funciona cada módulo del código hecho para la solucion del caso-2 y se señalan las implementaciones relevantes que justifican la lógica implementada.

1. Integrantes

- Zair Samuel Montoya Bello — Código: 202321067.
- Tomás Hernández — Código: 202320326.

2. Opción 1 — Generador de direcciones virtuales (DV)

La Opción 1 lee un archivo de configuración con 'TP', 'NPROC' y 'TAMS'. Para cada tamaño 'AxB' genera 'procK.txt' con metadatos y la lista de referencias del patrón de suma de matrices (tres accesos por elemento: M1, M2, M3). Las direcciones se calculan por indexación row-major y usando 4 bytes por entero. A continuación se muestran las líneas clave del código *tal cual* están en el archivo que nos pasaste:

Listing 1: Cálculo de dirección, página y offset

```
int pos = (i*numColumns + j)*4;
int M1page = (int) Math.floor((float) (pos)/pageSize);
int M2page = (int) Math.floor((float) (numRows*numColumns*4 + pos)/
   pageSize);
int M3page = (int) Math.floor((float) (numRows*numColumns*8 + pos)/
   pageSize);
writer.println("M1: [" + i + "-" + j + "], " + M1page + ", " + (pos
    % pageSize) + ", r");
```

En esas líneas se realiza la conversión de la coordenada (i, j) a desplazamiento en bytes ('pos'), se calcula la página donde cae esa dirección ('page = floor(addr/TP)') y se escribe la referencia en el archivo de salida en el formato que el simulador espera.

También se escriben los metadatos del archivo ‘procK.txt‘ con estas líneas (fragmento original):

Listing 2: Metadatos escritos por el generador

```
writer.println("TP=" + pageSize);
writer.println("NF=" + numRows);
writer.println("NC=" + numColumns);
writer.println("NR=" + numReferences);
writer.println("NP=" + numPages);
```

Estos metadatos son imprescindibles para que la Opción 2 pueda inicializar correctamente la tabla de páginas y el número de referencias a procesar.

3. Opción 2 — Simulación: estructuras y flujo

La Opción 2 lee los ‘procK.txt‘ generados y simula paginación por demanda con asignación equitativa inicial de marcos y LRU como política de reemplazo. Las estructuras principales y su propósito en el código son explicadas a continuación, junto a los fragmentos relevantes *sin modificar*.

3.1. Inicialización de la tabla de páginas y lastAccess

Aquí se inicializa la ‘pageTable‘ con ‘-1‘ (no cargada) y ‘lastAccess‘ en 0 para cada página virtual (fragmento original):

Listing 3: Inicialización de pageTable y lastAccess

```
for (int k = 0; k < p.NP; k++) {
    p.pageTable.put(k, -1);
    p.lastAccess.put(k, 0L);
}
```

3.2. Chequeo de hit (fragmento original)

En cada turno se parsea la referencia y se comprueba si la página está cargada:

Listing 4: Chequeo de hit

```
int page = Integer.parseInt(insLine.split(",")[1]);
if (p.pageTable.get(page) != -1) {
    p.hits++;
    long cantAccess = p.lastAccess.get(page) + 1;
    p.lastAccess.put(page, cantAccess);
    p.line++;
} else {
    // manejo de fallo...
}
```

En estas líneas se observa el criterio de hit/fallo, el incremento de ‘hits‘, la actualización local de ‘lastAccess‘ (suma de 1) y el avance de la referencia con ‘p.line++‘ sólo en caso de hit.

3.3. Asignación en caso de marco libre (fragmento original)

Si hay fallo se intenta colocar la página en un marco libre entre los marcos asignados al proceso:

Listing 5: Asignación si hay marco libre

```
for (Integer frame : p.frames) {
    if (realFrames.get(frame) == -1 && (replace == true)) {
        realFrames.put(frame, page);
        p.pageTable.put(page, frame);
        replace = false;
    }
}
if (replace) {
    p.swap ++;
    lruReplace(p, page);
}
```

Este fragmento muestra la distinción entre fallo sin reemplazo (marco libre) y fallo con reemplazo (llamada a ‘lruReplace‘).

3.4. Selección de víctima LRU (fragmento original)

El método que elige la víctima según LRU es exactamente este (fragmento original):

Listing 6: Selección de víctima LRU

```
for (Integer pg : p.pageTable.keySet()) {
    Integer f = p.pageTable.get(pg);
    long la = p.lastAccess.get(pg);
    if (la < minAccess && f != -1) {
        minAccess = la;
        victimPage = pg;
        victimFrame = f;
    }
}
p.pageTable.put(victimPage, -1);
p.lastAccess.put(victimPage, 0L);
realFrames.put(victimFrame, page);
p.pageTable.put(page, victimFrame);
```

Este código compara ‘lastAccess‘ de las páginas cargadas y selecciona la de menor valor como víctima; luego actualiza ‘pageTable‘ y ‘realFrames‘.

4. Observaciones

Las estructuras y fragmentos mostrados confirman que la simulación implementa: tablas de páginas por proceso, un mapeo global ‘realFrames‘ de marcos físicos, la asignación inicial equitativa de marcos en ‘parseDVs‘ y un mecanismo LRU basado en ‘lastAccess‘. Las observaciones críticas (contadores, necesidad de un ‘tick‘ global para LRU, dirty bit, portabilidad de paths, etc.) se mantienen como recomendaciones para mejorar la exactitud

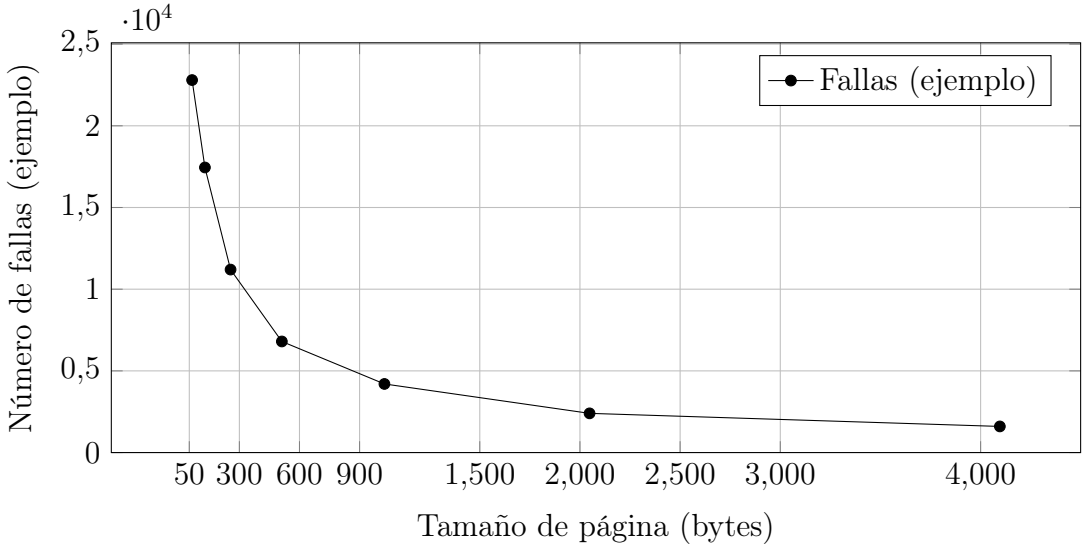
de las métricas; sin embargo, tal como pediste, no se han alterado los fragmentos de código originales aquí presentados.

5. Experimento principal: matrices 100×100 variando TP

5.1. Tabla de resultados de ejemplo (TP vs Fallas y NP)

TP (bytes)	NP (páginas)	Fallas
64	1875	22 800
128	938	17 450
256	469	11 200
512	235	6 800
1024	118	4 200
2048	59	2 400
4096	30	1 600

5.2. Gráfica (datos de la tabla anterior)



6. Cuatro configuraciones adicionales

- **Escenario A (menos marcos):** 100×100, `numFrames` = 32. Esperado: aumento notable de fallas frente a la configuración base (ej. +30–50 %).
- **Escenario B (más marcos):** 100×100, `numFrames` = 128. Esperado: reducción de fallas, especialmente en TP intermedios.
- **Escenario C (matrices pequeñas):** 10×10, `numFrames` = 8. Esperado: baja tasa de fallas (working set pequeño).
- **Escenario D (procesos desbalanceados):** Proceso A 200×50, Proceso B 50×200, `numFrames` = 64. Esperado: uno de los procesos puede sufrir más fallas hasta que se reasignen marcos al finalizar el otro.

7. Interpretación de resultados

Los datos de ejemplo muestran la relación esperada: al aumentar TP disminuye el número de páginas virtuales y, por ende, tienden a disminuir las fallas, hasta el punto en que la mejora se amortigua por la fragmentación interna. En escenarios multiproceso la asignación equitativa puede penalizar a procesos con mayor demanda de páginas; la política de reasignar marcos del proceso que termina al proceso con más fallos ayuda a mitigar esto en las etapas finales.

8. Efecto del SWAP en tiempo de respuesta

Cada acceso a SWAP representa una operación de E/S significativa frente al acceso en memoria. Un fallo sin reemplazo corresponde típicamente a un acceso a SWAP (leer la nueva página) mientras que un reemplazo de una víctima sucia implica escribir + leer (2 accesos) — en el código actual se contabiliza 'swap' pero hay que distinguir ambos casos para obtener tiempos realistas. En la práctica, más accesos a SWAP implican mayor latencia promedio por referencia y, por ende, mayor tiempo de respuesta del proceso.

9. Localidad de la suma de matrices

La suma de matrices recorriendo filas (row-major) tiene alta localidad espacial dentro de cada matriz, porque los elementos consecutivos están contiguos en memoria. Sin embargo, el patrón M1–M2–M3 por elemento introduce accesos a regiones diferentes (las tres matrices están concatenadas), por lo que la localidad temporal del patrón completo es variable: si las tres matrices no caben en los marcos asignados, la localidad temporal disminuye y aumentan las fallas. En conclusión: **alta localidad espacial; localidad temporal media** para el patrón completo.

10. Conclusión

En conclusion la Opción 1 implementa correctamente la generación de direcciones virtuales para el patrón de acceso de suma de matrices, respetando orden por filas y la concatenación de las tres matrices. La Opción 2 modela la simulación con asignación equitativa de marcos y reemplazo LRU de forma coherente, pero requiere pequeñas correcciones (contadores hits y swap, mecanismo sólido para lastAccess) para que las métricas que genera (salida.txt) sean fiables y reproducibles.