Caso 2 – Infraestructura Computacional

Nicolas Arango - 202220342
Samuel Hernández – 202213772

Memoria Virtual

Problemática Planteada

La problemática central radica en cómo simular y gestionar el sistema de paginación de un sistema operativo para entender y optimizar su comportamiento. Específicamente, el caso incluye cómo representar la memoria física (RAM) y la memoria virtual, cómo manejar el acceso a páginas de memoria que pueden o no estar cargadas en RAM, y cómo implementar políticas de reemplazo de páginas para gestionar eficientemente la memoria limitada disponible. Además, el sistema debe ser capaz de simular el comportamiento de lectura de páginas desde la memoria virtual (SWAP) cuando se encuentran fallos de página y el posterior cálculo de métricas de rendimiento, como el número de fallos de página y el porcentaje de hits.

Solución Propuesta

La solución propuesta para el caso se desarrolla mediante la implementación de un simulador de paginación, que utiliza estructuras de datos específicas para modelar los componentes clave del sistema de paginación:

Memoria RAM: Se representa como una lista sincronizada de objetos Pagina, cada uno conteniendo información sobre una página específica cargada en la memoria física.
Memoria Virtual y Cola de Referencias: Una cola concurrente almacena referencias pendientes a páginas que deben ser accedidas, simulando solicitudes de acceso a la memoria virtual.
Políticas de Reemplazo de Páginas: Se implementan políticas como FIFO o LRU mediante mecanismos adicionales dentro del simulador, permitiendo la selección inteligente de qué páginas reemplazar cuando la RAM está llena.
Procesamiento de Referencias y Simulación de SWAP: Un hilo dedicado consume referencias de la cola, simulando el acceso a páginas, el manejo de fallos de página y la actualización de estructuras de datos basado en las políticas de reemplazo.

Algoritmo para generar referencias:

```java
public static void generarArchivoDeReferencias(String nombreArchivo, int tp, int nf, int nc) {
    int tamFiltro = 3; // El tamaño del filtro es 3x3
    int tamElemento = 4; // Cada elemento (int) ocupa 4 bytes
    int elementosFiltro = tamFiltro * tamFiltro; // 9 elementos en el filtro
    int np = (int) Math.ceil((double) (elementosFiltro + nf * nc * 2) * tamElemento / tp);
    int nr = nf * nc * 3; // Cada elemento de M y R genera una referencia, cada elemento de F se usa nf*nc
                          // veces

    try (PrintWriter writer = new PrintWriter(nombreArchivo)) {
        writer.printf(format:"TP=%d\nNF=%d\nNC=%d\nNF_NC_Filtro=%d\nNR=%d\nNP=%d\n", tp, nf, nc, tamFiltro, nr, np);

        int offset = 0; // Desplazamiento inicial para el filtro
        for (int i = 0; i < nf; i++) {
            for (int j = 0; j < nc; j++) {
                // Referencias para la matriz de datos (M)
                int paginaM = (elementosFiltro * tamElemento + (i * nc + j) * tamElemento) / tp;
                int desplazamientoM = (elementosFiltro * tamElemento + (i * nc + j) * tamElemento) % tp;
                writer.printf(format:"M[%d][%d],%d,%d,R\n", i, j, paginaM, desplazamientoM);

                // Usamos el filtro (F) para cada elemento de M, asumiendo reutilización
                // completa del filtro
                for (int fi = 0; fi < tamFiltro; fi++) {
                    for (int fj = 0; fj < tamFiltro; fj++) {
                        int paginaF = ((fi * tamFiltro + fj) * tamElemento) / tp;
                        int desplazamientoF = ((fi * tamFiltro + fj) * tamElemento) % tp;
                        writer.printf(format:"F[%d][%d],%d,%d,R\n", fi, fj, paginaF, desplazamientoF);
                    }
                }

                // Referencias para la matriz de resultados (R), que sigue a M en memoria
                int paginaR = ((elementosFiltro + nf * nc) * tamElemento + (i * nc + j) * tamElemento) / tp;
                int desplazamientoR = ((elementosFiltro + nf * nc) * tamElemento + (i * nc + j) * tamElemento) % tp;
                writer.printf(format:"R[%d][%d],%d,%d,W\n", i, j, paginaR, desplazamientoR);
            }
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

tp, nf y nc son valores ingresados por el usuario, que representan el tamaño de página, y de la matriz. Los valores al inicio se basan en lo dado por el enunciado, donde el filtro es 3x3, y cada elemento ocupa 4 bytes.

El writer se encarga de redactar el txt resultante. Lo primero es escribir los datos básicos (la cabecera) sobre la generación.

Luego, está el cómo se generan las referencias:

Se tiene un doble for al inicio para recorrer la matriz de datos. Para cada posición (i,j), se halla en que página debería estar y su desplazamiento (se escribe en el archivo).

Para el filtro, se itera este sobre cada elemento M, hallando a su vez su posición y desplazamiento.

Por último, se crea la referencia del elemento.

*Se usa un bloque try-catch para el manejo del archivo

• 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).

1. Lista de Pagina (RAM)

Se utiliza una lista (List<Pagina>) para representar la memoria física (RAM). Cada elemento de esta lista representa una página cargada en la RAM, donde Pagina es una clase que al menos contiene un identificador de número de página y el bit R (bit de referencia).

Cómo y Cuándo se Actualiza:

Carga de Páginas: Cuando se accede a una página que no está en RAM (fallo de página), se crea una nueva instancia de Pagina y se añade a esta lista, simulando la carga de la página en la memoria física.
Reemplazo de Páginas: Si se accede a una página que no está en RAM y la RAM ya está llena (alcanzado su capacidad máxima), se selecciona una página existente en la lista para ser reemplazada. La selección depende de la política de reemplazo implementada (por ejemplo, FIFO, LRU). La página seleccionada se elimina de la lista, y la nueva página se añade.
Actualización del Bit R: Periodicamente, el bit R de las páginas en esta lista puede ser reseteado o actualizado, dependiendo del reemplazo de páginas y del algoritmo que gestiona el bit R.

2. Cola de Referencias (Queue<ReferenciaPagina>)
Se emplea una cola concurrente (ConcurrentLinkedQueue<ReferenciaPagina>) para almacenar las referencias de páginas pendientes de procesar. ReferenciaPagina es una clase que al menos contiene el número de página y el tipo de acceso (lectura o escritura).

Cómo y Cuándo se Actualiza:

Añadir Referencias: Las referencias se añaden a esta cola cuando se leen desde el archivo de referencias o se generan durante la simulación. Cada referencia representa un acceso futuro a una página específica en la memoria virtual.
Procesamiento de Referencias: Un hilo específico (por ejemplo, ActualizadorPaginas) consume referencias de esta cola, las procesa una por una, y realiza las acciones necesarias, como verificar si la página está en RAM y manejar fallos de página.

3. Implementación de Políticas de Reemplazo
Para implementar políticas de reemplazo de páginas como LRU o FIFO, se pueden usar estructuras adicionales como mapas (Map) para LRU, donde se almacena el momento del último acceso a cada página, o se puede confiar en el orden de las páginas en la lista de RAM para FIFO.

Cómo y Cuándo se Actualiza:

LRU (Least Recently Used): Para cada acceso a una página en RAM, se actualiza su momento de último acceso en el mapa. Cuando es necesario reemplazar una página, se elige la que tiene el momento de último acceso más antiguo.
FIFO (First In, First Out): La lista de RAM inherentemente sigue el orden FIFO para el reemplazo de páginas si simplemente añade nuevas páginas al final y elimina páginas del principio cuando se requiere hacer espacio.

Esquema de sincronización:
La sincronización está en el objeto SimuladorPaginacion. El thread ActualizadorBitR sincroniza sobre ese objeto.

```java
@Override
public void run() {
    try {
        while (!Thread.currentThread().isInterrupted()) {
            synchronized (simulador) {
                List<Pagina> ram = simulador.getRam();
                for (Pagina pagina : ram) {
                    pagina.bitR = false; // Resetea el bit R de cada página.
                }
            }
            Thread.sleep(intervalo);
        }
```

*Extraído de ActualizadorBitR

```java
public synchronized void procesarReferencia(int numeroPagina, char tipoAcceso) {
```

*Extraído de SimuladorPaginacion

Ambos escriben en la RAM (específicamente leen y modifican las páginas). Por lo que no pueden acceder a la misma vez a esta.

Mientras se actualiza el bit R, no se puede procesar referencias porque se podrían sobrescribir el acceso a una página. En cuanto se libere, el simulador podrá procesar sin sufrir el riesgo de sobrescribir.

Tabla con hits y misses en varios casos simulados:

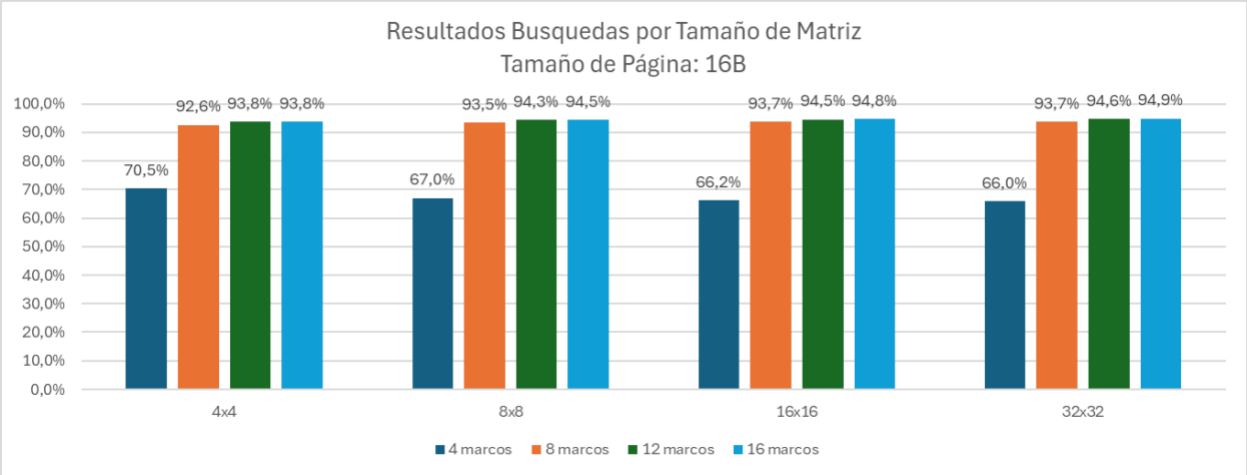
Se tuvo en cuenta dos tamaños de página: 16B y 32B. Para cada uno, usaron matrices de tamaño: 4, 8, 16 y 32.

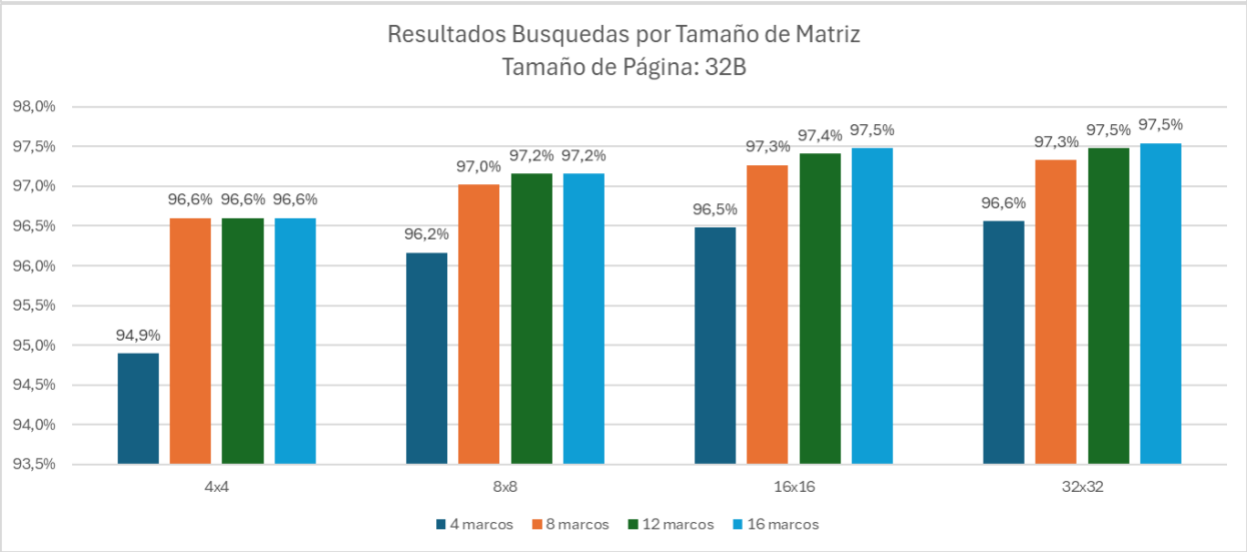
Paginas de 16B, matriz 4x4			
Marcos Asignados	Total Referencias	Hits	Fallas
4	176	124	52
8	176	163	13
12	176	165	11
16	176	165	11
Paginas de 16B, matriz 8x8			
Marcos Asignados	Total Referencias	Hits	Fallas
4	704	472	232
8	704	658	46
12	704	664	40
16	704	665	39
Paginas de 16B, matriz 16x16			
Marcos Asignados	Total Referencias	Hits	Fallas
4	2816	1864	952
8	2816	2638	178
12	2816	2662	154
16	2816	2669	147
Paginas de 16B, matriz 32x32			
Marcos Asignados	Total Referencias	Hits	Fallas
4	11264	7432	3832
8	11264	10558	706
12	11264	10654	610
16	11264	10685	579

Paginas de 32B, matriz 4x4			
Marcos Asignados	Total Referencias	Hits	Fallas
4	176	167	9
8	176	170	6
12	176	170	6
16	176	170	6
Paginas de 32B, matriz 8x8			
Marcos Asignados	Total Referencias	Hits	Fallas
4	704	677	27
8	704	683	21
12	704	684	20
16	704	684	20
Paginas de 32B, matriz 16x16			
Marcos Asignados	Total Referencias	Hits	Fallas
4	2816	2717	99
8	2816	2739	77
12	2816	2743	73
16	2816	2745	71
Paginas de 32B, matriz 32x32			
Marcos Asignados	Total Referencias	Hits	Fallas
4	11264	10877	387
8	11264	10963	301
12	11264	10980	284
16	11264	10987	277

Gráficas:

Se grafican las anteriores tablas, con el porcentaje de aciertos (hits/total referencias) vs el marco (RAM), agrupados por el tamaño de la matriz.





Analisis: Se puede observar que la mejor manera de aumentar el porcentaje de éxito es aumentar la RAM. Esto tiene sentido porque habrá el espacio para tener más páginas cargadas a la vez, derivando en que se disminuya la frecuencia con que se reemplazan páginas. Pasar de un marco de 4 a 8 aumenta considerablemente los aciertos (esto se bien en 16B, donde se pasa del orden de 70% a 94%); al seguir aumentando el marco se sigue mejorando el porcentaje, pero no de gran manera (se estanca en 94%).