Caso 2
Nicolas Arango - 202220342
Samuel Hernández - 202213772

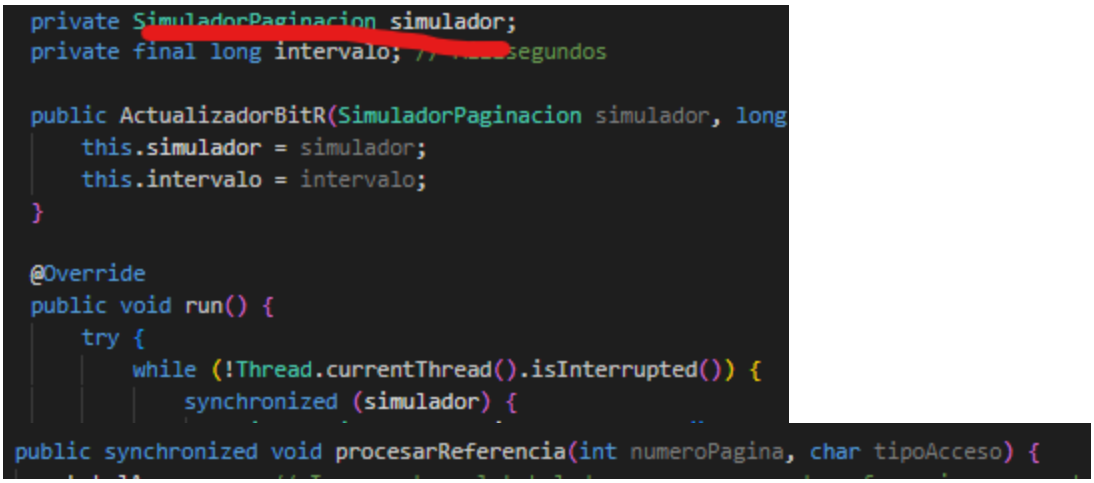
Algoritmo para generar referencias:
Se usa el siguiente procedimiento, teniendo en cuenta que tp, nf y nc son ingresados por el usuario

```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();
    }
}
```

El writer se encarga de redactar el txt resultante. Lo primero es escribir los datos básicos 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, se halla en que página debería estar y su desplazamiento (se escribe en el archivo).
Ocurre lo mismo para las otras dos matrices (filtro y resultado)

• 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).

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



```
private SimuladorPaginacion simulador;
private final long intervalo; // ...segundos

public ActualizadorBitR(SimuladorPaginacion simulador, long
    this.simulador = simulador;
    this.intervalo = intervalo;
}

@Override
public void run() {
    try {
        while (!Thread.currentThread().isInterrupted()) {
            synchronized (simulador) {
public synchronized void procesarReferencia(int numeroPagina, char tipoAcceso) {
```

El otro punto donde se sincroniza es en el método procesarReferencia del simulador. Estos procesos no se pueden hacer a la vez.

Tabla con hits y misses en varios casos simulados:

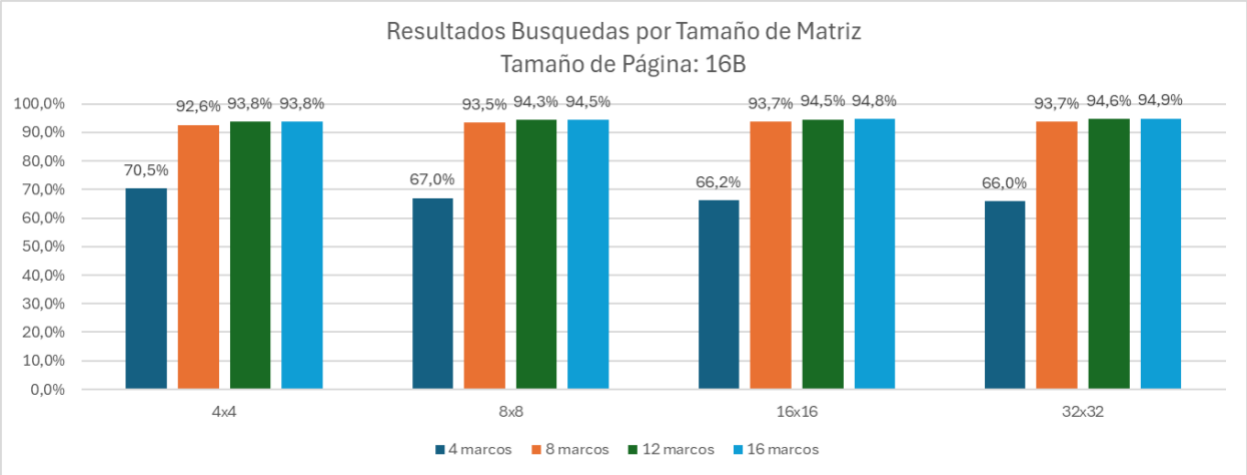
Se usaron dos casos, uno con tamaño de página 16B y otro con 32B. Dentro de cada uno, se tienen matrices de 4, 8, 16 y 32.

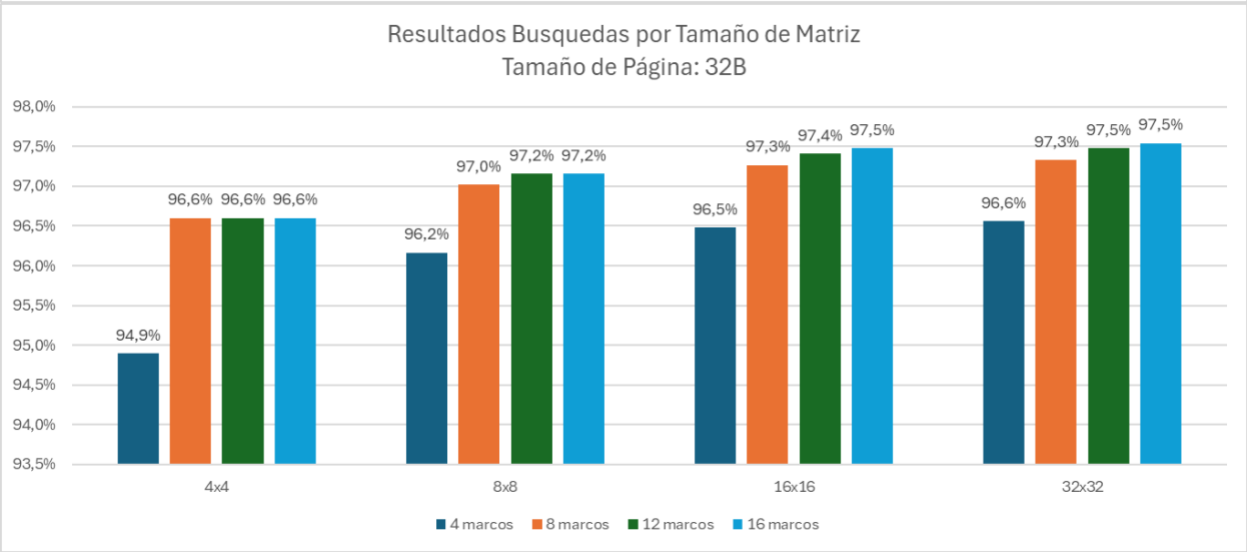
Paginas de 16B, matriz 4x4			
Marcos Asignados	Total Referencias	Hits	Fallas
4	176	124	52
8	176	163	13
12	176	165	11
16	176	165	11
Paginas de 16B, matriz 8x8			
Marcos Asignados	Total Referencias	Hits	Fallas
4	704	472	232
8	704	658	46
12	704	664	40
16	704	665	39
Paginas de 16B, matriz 16x16			
Marcos 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, pero con el porcentaje de aciertos (hits/total referencias)





Analisis: Se puede observar que la mejor manera de aumentar el porcentaje de éxito es aumentar la RAM. Esto se debe a que habrá más espacio para tener muchas páginas cargadas a la vez. El desempeño afecta y recomienda mayor si el marco es pequeño, pero a partir del marco 8 no se refleja un aumento muy pronunciado. Aumentar el tamaño de la matriz también mejora la ratio, pero no de manera significativa.