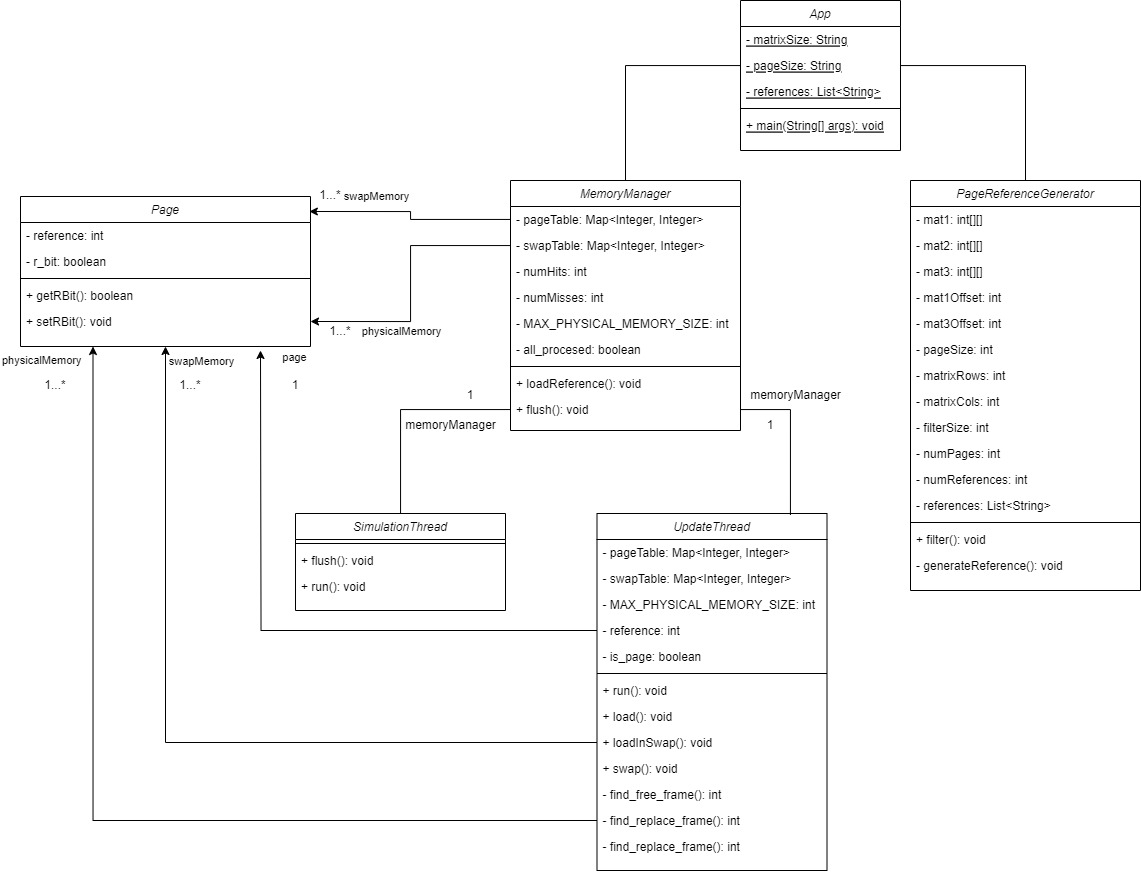
Caso #2

Integrantes

* Henry Santiago Antolinez Gentil – 202121785
* Santiago Celis Rengifo – 202111131
* Diego Alejandro Pulido Bonilla – 202215711

Diagrama de clases



Descripción del algoritmo usado para generar las referencias de página

El método acepta cuatro parámetros: matrix (la matriz a la que se accede), row y col (las coordenadas de la posición en la matriz) y operation (una cadena que indica si la operación es de lectura o escritura). Dentro del método se definen dos variables: matrixOffset y matCol. La primera (matrixOffset) representa el desplazamiento inicial de la matriz correspondiente dentro de la memoria virtual, mientras que matCol representa el número de columnas de la matriz. Estos valores se asignan según la matriz (mat1, mat2 o mat3) a la que se accede.

Para mat1 (matriz de datos), matrixOffset es igual al espacio ocupado por la matriz filtro (ya que esta se almacena primero), y matCol es igual a matrixCols (el número de columnas de la matriz de datos). Para la matriz mat2 (filtro), matrixOffset es 0 (ya que el filtro se coloca al inicio de la memoria virtual), y matCol es 3 (debido a que el filtro tiene dimensiones 3x3). Para mat3 (matriz de resultados), matrixOffset es la suma del espacio ocupado por la matriz filtro con el de la matriz resultado, y matCol es igual a matrixCols (el número de columnas de la matriz de datos, ya que la matriz de resultados tiene las mismas dimensiones que la matriz de datos).

Luego, se calcula el desplazamiento total dentro de la memoria virtual para la posición (row, col) de la matriz correspondiente. Esto se hace sumando matrixOffset y el desplazamiento dentro de la matriz para esa posición (calculado como row \* matCol + col, multiplicado por 4 bytes, asumiendo que cada elemento ocupa 4 bytes).

Finalmente se calcula el número de página virtual (pageNum) dividiendo el desplazamiento total entre el tamaño de página (pageSize) y el desplazamiento dentro de la página virtual (offsetInPage) se calcula tomando el residuo del desplazamiento dividido por pageSize. Ya por último, se genera la referencia en el formato solicitado, se añade a la lista de referencias y se suma uno al contador de referencias.

Descripción de las estructuras de datos usadas para simular el comportamiento del sistema de paginación y cómo usa dichas estructuras

Para poder simular el comportamiento del sistema de paginación se utilizaron las siguientes estructuras de datos:

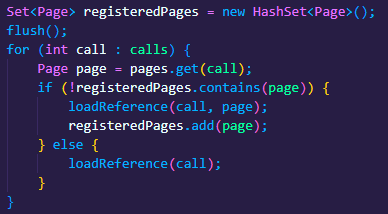
Listas (ArrayList): Tanto para modelar la memoria física como para modelar la memoria Swap se empleó una lista de páginas (una para cada una). La primera lista almacena las páginas que están actualmente cargadas en la memoria física y la segunda, simula el almacenamiento de páginas que han sido removidas de la memoria física (RAM).



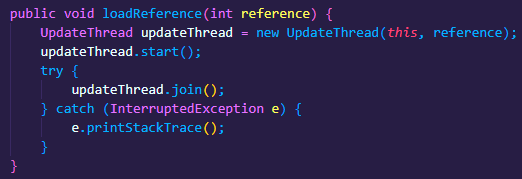
Mapas (HashMap): Se crearon mapas para modelar las tablas de referencias de la memoria física y de la Swap. En estos mapas se tiene una llave-valor que representa el número de una página y su respectiva referencia a la ubicación en donde se encuentra. De esta manera, se simula la forma en la que el SO guarda la información de las páginas para realizar traducciones de direcciones virtuales a físicas.



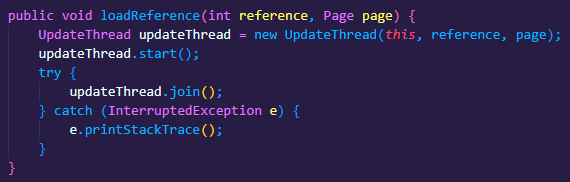
Conjunto (Set): Otra estructura de datos importante que manejamos es el conjunto de páginas que se tienen. Este set nos permite llevar un registro de las páginas que han sido referenciadas durante el proceso de creación del constructor de ‘MemoryManager’. El siguiente código se tiene para que cada página se cargue solo una vez en el proceso de inicialización (si ya estaba cargada solo se carga el llamado a la misma, más no la página en sí):



En el anterior código, se tienen dos casos, si la página ya había sido referenciada o no, para el primer caso solo se carga el llamado de acceso a esa página con el método ‘loadReference(call)’. Este método crea un nuevo hilo y realiza la carga de la referencia a una página en la memoria física:



Por otro lado, si la página no ha sido referenciada antes, se añade a al set de páginas registradas y se llama al segundo método de ‘loadReference(call, page)’. Este método crea un hilo para cargar la página y la referencia o acceso (call) a esta.



Esquema de sincronización usado

Para la sincronización de los Threads, además de sincronizar todos los métodos que modificaban o leían los arreglos y mapas que representaban la memoria física y el swap se usaron los siguientes patrones de concurrencia.

* Patrón monitor: Dado que se decidió mantener los métodos dentro de las clases de los Threads, y debido a que por la lógica del programa los Threads ya tenían una referencia de la clase principal, se decidió que la sincronización se iba a hacer con respecto a dicho objeto (memoryManager), pues esto permitiría tener sincronización entre los dos hilos que realizaban tareas sobre los datos.
* Dado que el hilo encargado del bit R tenía que ejecutarse recurrentemente cada 4ms este se implementa bajo una espera semi-activa con un sleep y una variable que hace de flag indicando si ya fueron procesadas todas las referencias.
* Finalmente se hizo uso del método join de los hilos en java que permite esperar a que un hilo finalice para que el otro pueda continuar, esto es importante ya que todos los hilos manejan los mismos espacios en memoria, con lo que es importante saber en qué momento un hilo dejó de modificar la memoria para que el otro pueda hacerlo.

Tabla con los datos recopilados

|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 16B, matriz 4x4** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 88 | 57 | 31 |
| 8 | 88 | 75 | 13 |
| 12 | 88 | 77 | 11 |
| 16 | 88 | 77 | 11 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 16B, matriz 6x6** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 324 | 195 | 129 |
| 8 | 324 | 256 | 68 |
| 12 | 324 | 276 | 48 |
| 16 | 324 | 293 | 31 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 16B, matriz 8x8** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 712 | 420 | 292 |
| 8 | 712 | 457 | 255 |
| 12 | 712 | 505 | 207 |
| 16 | 712 | 563 | 149 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 32B, matriz 4x4** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 88 | 80 | 8 |
| 8 | 88 | 82 | 6 |
| 12 | 88 | 82 | 6 |
| 16 | 88 | 82 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 32B, matriz 6x6** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 324 | 264 | 60 |
| 8 | 324 | 306 | 18 |
| 12 | 324 | 313 | 11 |
| 16 | 324 | 313 | 11 |

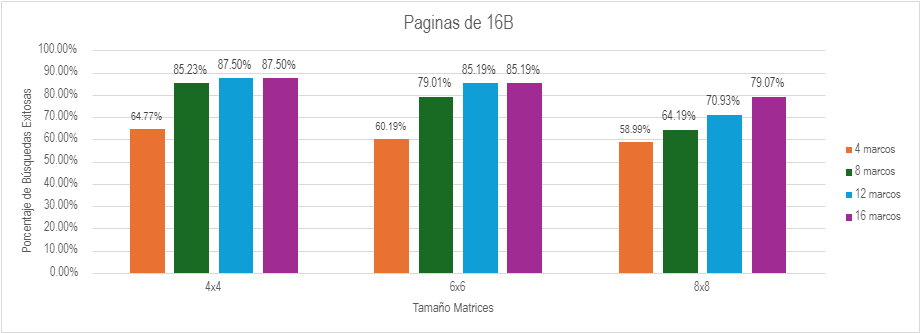
|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 32B, matriz 8x8** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 712 | 534 | 178 |
| 8 | 712 | 630 | 82 |
| 12 | 712 | 676 | 36 |
| 16 | 712 | 693 | 19 |

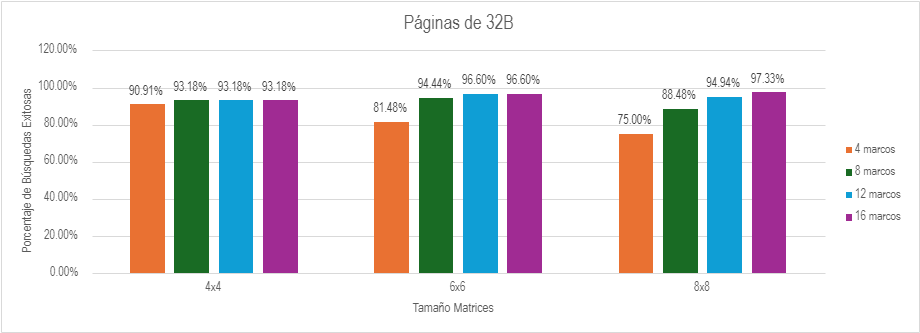
|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 64B, matriz 4x4** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 88 | 85 | 3 |
| 8 | 88 | 85 | 3 |
| 12 | 88 | 85 | 3 |
| 16 | 88 | 85 | 3 |

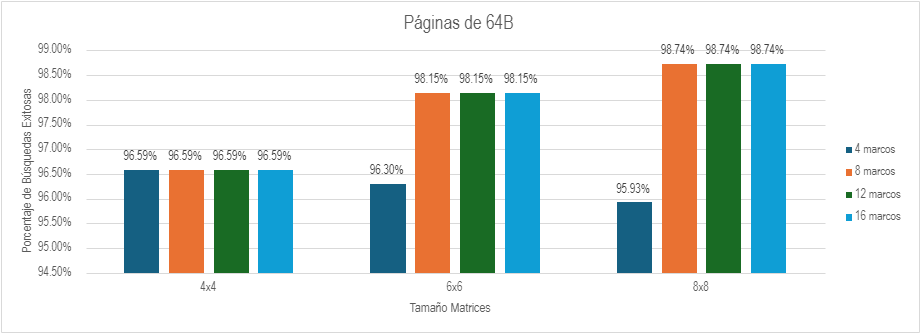
|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de 64B, matriz 6x6** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 324 | 312 | 12 |
| 8 | 324 | 318 | 6 |
| 12 | 324 | 318 | 6 |
| 16 | 324 | 318 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Páginas de64B, matriz 8x8** | | | |
| **Marcos Asignados** | **Total Referencias** | **Hits** | **Fallas** |
| 4 | 712 | 683 | 29 |
| 8 | 712 | 703 | 9 |
| 12 | 712 | 703 | 9 |
| 16 | 712 | 703 | 9 |

Serie de gráficas







Interpretación de los resultados

Dado que inicialmente los datos se encuentran en disco, el porcentaje máximo de búsquedas exitosas nunca será del 100%, si no que va a ser relativo de cuantas referencias diferentes se necesiten, pues estas tendrán que salir en un principio de la memoria persistente, lo cual generará el 100% de las veces fallos de memoria. Teniendo en cuenta esto, el algoritmo se comporta de manera esperada, pues se puede evidenciar una correlación entre el porcentaje de éxitos y el número de marcos. Adicionalmente algo para tener en cuenta es que para cada fila o columna que se agrega a la matriz, los registros aproximadamente se duplican con respecto a el tamaño anterior (e.g para la matriz 4x4 los registros son 88 y para la de 8x8 el número sube hasta 712), por lo que el algoritmo tiene un buen rendimiento considerando que la caída del porcentaje con muchos más registros no es problemática e incluso con un ligero aumento de los marcos de página el porcentaje sube a niveles por encima del 80%.

Además, también es posible observar que a medida que el tamaño de las páginas aumenta, el porcentaje de búsquedas exitosas (hits) también se incrementa. Esto se debe a que, con páginas de mayor tamaño, se puede alojar más datos en la memoria principal, reduciendo la necesidad de acceder a la memoria secundaria (disco). Cuando se trabaja con matrices más grandes (8x8), se requiere más espacio en memoria, y las páginas de mayor tamaño pueden contener una mayor cantidad de elementos de la matriz, lo que resulta en un menor número de fallas de página y, por lo tanto, un porcentaje más alto de búsquedas exitosas.