Santiago Rodriguez202011182

Luis Felipe Dussan 201912308

Kevi David Alvarez Romero -

202022834

Caso 2: Memoria Virtual

Contenido

Introducción

El propósito del caso era crear una simulación que demostrara cómo un sistema operativo administra la memoria RAM de un dispositivo mediante el uso de memoria virtual y un sistema de paginación. Se utilizó un programa escrito en Java para simular el sistema de paginación y el algoritmo de envejecimiento, y se observó cómo el sistema de paginación toma decisiones de reemplazo de página mientras se ejecuta un proceso (suma de dos matrices). En resumen, la simulación demostró cómo funciona el sistema de paginación y cómo toma decisiones para administrar la memoria de manera eficiente.

Funcionamiento del Sistema (general)

Se implemento Main como nuestra clase principal, sus responsabilidades constan de manejar todo lo que la interfaz de usuario se refiere, principalmente la carga de archivos de configuración). Nuestra clase main se podrá ver como la implementación del sistema operativo.

La clase de Modo 1 donde se encuentra la implementación de la opción 1, ya que es donde se genera la tabla de páginas y la tabla de referencias a dichas paginas dependiendo de parámetros dados por el usuario (tamaño de página, tamaño de las matrices, etc). A su vez clase como Info son auxiliares a la clase modo 1, y estas se usan para crear la tabla de referencias que produce esta clase. Internamente la clase modo1 crea un arraylist con las referencias que facilitan el eventual uso de estas en el modo 2.

La clase Modo2 se encarga de coordinar los procesos necesarios para simular el sistema de paginación junto con el algoritmo de envejecimiento. Para ello, utiliza tres clases auxiliares: Thread1, Thread2 y Buffer.

La clase Thread1 accede a la tabla de referencias que contiene el orden en que se acceden las páginas. Su tarea es acceder al buffer para modificar la tabla del marco de página, que contiene las páginas accedidas y su "edad". Puede agregar una nueva página, actualizar su edad o generar un fallo de página y reemplazar la página más antigua con la actual. Después de realizar estas acciones, el Thread1 se duerme durante dos milisegundos.

La clase Thread2 se encarga de ejecutar el algoritmo de envejecimiento. Accede al buffer para actualizar la edad de cada página en la tabla del marco de página. Después de ejecutar el envejecimiento, Thread2 se duerme durante un milisegundo. Ambos threads se ejecutan simultáneamente hasta que el Thread1 termine de recorrer la tabla de referencias.

La clase Buffer contiene la tabla del marco de página y está sincronizada en relación a ella. Solo permite que un thread acceda a ella a la vez mediante un método sincronizado.

Funcionamiento opción 1

Algoritmo de generación de referencias de pagina

Dentro del Proyecto, el modo 1 y todos sus requerimientos están implementados en la clase de modo1.java. A una instancia de esta clase se le son pasados todos los parámetros de ejecución de la clase correspondientes a (tamaño de página, numero de filas, numero de columnas ,etc.). Como vemos en las siguientes ilustraciones sobre como son visualizadas estas matrices al ser almacenadas en un sistema de memoria virtual.

Matriz 1

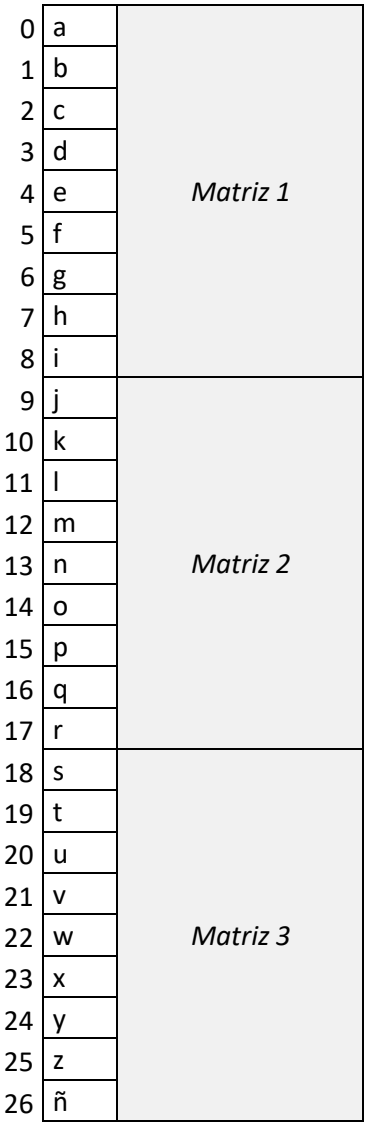
a	b	c
d	e	f
g	h	i

Matriz 2

j	k	l
m	n	o
p	q	r

s	t	u
v	w	x
y	z	ñ

Matriz 3



Al ser creada una instancia lo primero que se hace es la creación de las tres matrices: matriz1, matriz2 y matriz3. Matriz 3 es creada como la suma de las otras dos matrices, pero la forma que se realiza esta suma es definida por el algoritmo del proyecto que en este caso corresponde a recorrido1. Mientras que se realiza esta suma se van guardando las referencias a los enteros de las matrices en el arreglo de virtualMemory, la cual representa la memoria virtual.

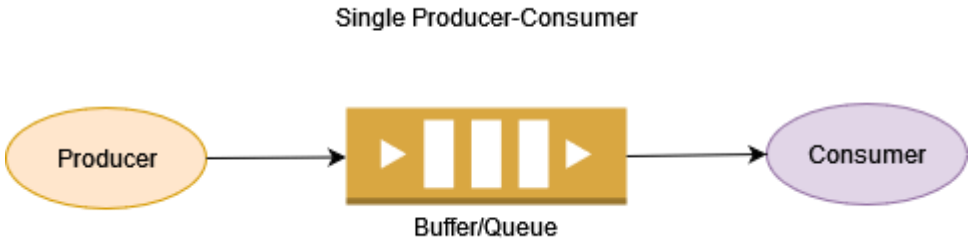
Respecto a la manera en que se calculo el sistema de paginación se hicieron distintas operaciones que nos permitieron calcular desplazamientos, referencias a paginas y los elementos que corresponderían a estas paginas. Esto se puede ver representado en el siguiente fragmento de código.

```java
public void comportamientoProceso() {

    int tamPagina= tamPag * tamEl;
    int tamMatriz = numFilas * numCol * tamEl;


    int pagMatriz = (tamMatriz + tamPagina- 1) / tamPagina;



    int tReferenumColias = numFilas * numCol * 3;


    StringBuilder resultado = new StringBuilder();

    // Agregar inumFilasormación básica al resultado
    resultado.append("TP=").append(tamPag).
    append("\n");
    resultado.append("NF=").append(numFilas).
    append("\n");
    resultado.append("NC=")
    .append(numCol).append("\n");
    resultado.append("NR=").append(tReferenumColias).
    append("\n");
```

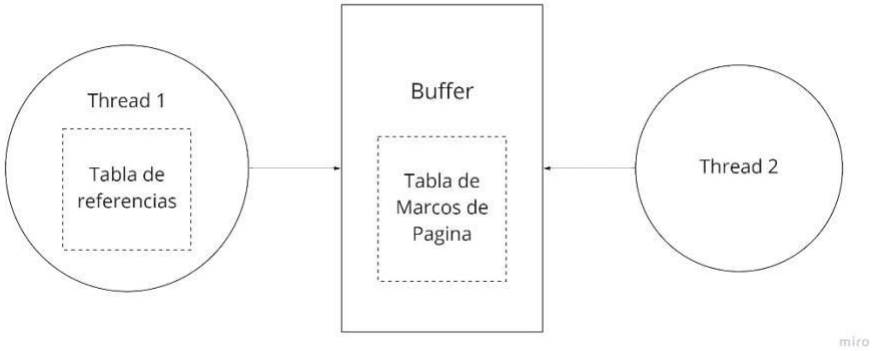
```java
    List<String> mA = new ArrayList<String>();
    List<String> mB = new ArrayList<String>();
    List<String> mC = new ArrayList<String>();

    for (int k = 0; k < 3; k++) {
        char matriz = (char) ('A' + k);
        // Iterar sobre cada fila de la matriz


        for (int i = 0; i < numFilas; i++) {
            int desplazamientoFila = (i % (tamPag / numCol)) * numCol;
            int pagVirtualFila = i / (tamPag / numCol);

            for (int x = 0; x < numCol; x++) {

                int paginaVirtualColumna = x / (tamPag / numFilas);
                int desplazamientoColumna = x % (tamPag / numFilas);
                int paginaVirtual = pagVirtualFila * marcos +
                paginaVirtualColumna + k * pagMatriz;
                int desplazamiento = (desplazamientoFila +
                desplazamientoColumna) * tamEl;
                String referenumColia = "[" + matriz + "-" + i + "-" + x + "]," + paginaVirtual
                + "," + desplazamiento + "\n";
                switch(matriz) {
                    case 'A':
                        mA.add(referenumColia);
                        break;
                    case 'B':
                        mB.add(referenumColia);
                        break;
                    case 'C':
                        mC.add(referenumColia);
                        break;
                }
            }
        }
    }
```

En estos fragmentos de código se ven representadas las operaciones necesarias para calcular las referencias hechas a paginas, teniendo en cuenta la información disponible en la literatura sobre la manera en que esta información es calculada.

Funcionamiento opción 2



En la opción 2 se utiliza una estructura de Producer-Consumer similar a la del caso 1. La clase Buffer almacena la tabla del marco de página, que contiene información sobre las páginas y su tiempo de última referencia en la tabla de referencias. El acceso a esta tabla está sincronizado y controlado por el monitor de la clase Buffer, permitiendo que solo un thread haga una operación en ella en un momento dado. La clase Thread1 actúa como un Productor/Consumidor, recorriendo la tabla de referencias y actualizando la tabla del marco de página con la última página accedida. Si la tabla del marco de página se llena, se produce un fallo de página y se tiene que realizar un swap. Después de actualizar la tabla del marco de página, el Thread 1 se va a dormir por 2 milisegundos antes de procesar la siguiente página. El Thread 2 es responsable de ejecutar el algoritmo de envejecimiento, que aumenta el tiempo de última referencia de cada página en la tabla del marco de página. Después de actualizar los valores, se duerme por 1 milisegundo antes de intentar ingresar a la tabla nuevamente. Este thread se ejecuta hasta que el Thread 1 ha procesado todas las referencias en la tabla de referencias



Es así con esta implementación que se puede representar el sistema de paginación y el uso del algoritmo de envejecimiento, al emplear la analogía del problema de Producer/Consumir.

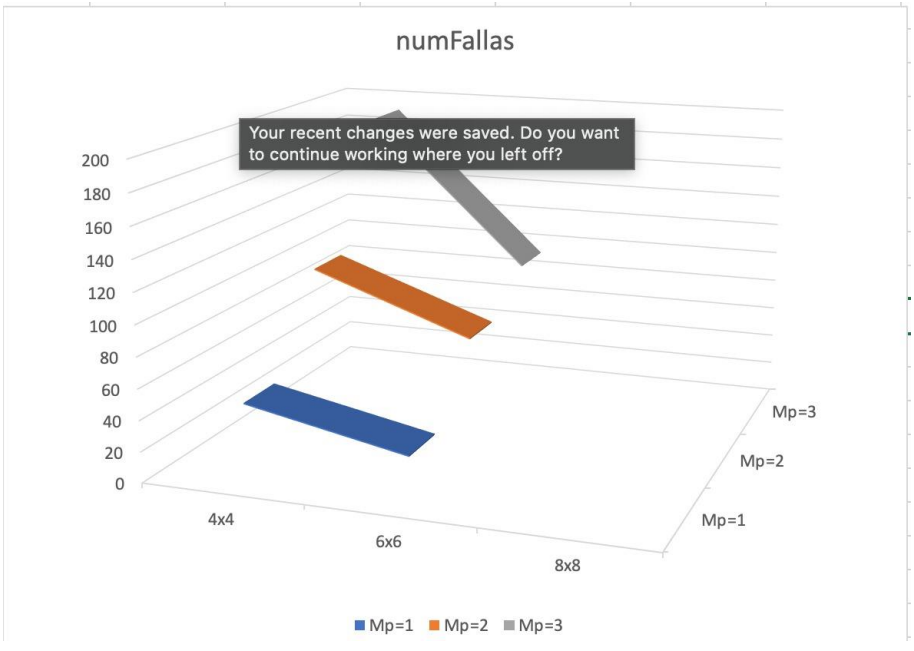
Sistema de sincronización

En este caso, se utilizó una estructura de listas de listas, específicamente ArrayList, para implementar el algoritmo en el que se encuentran involucrados el Thread 1 y Thread 2. Se establecieron ciertos requisitos para leer una referencia de página, como la disponibilidad de espacio en los marcos de página y al menos una página en los marcos de página. La sincronización se basó en la precisión de los intervalos de tiempo, permitiendo que cada 2 milisegundos se leyera una referencia de página o se produjera un fallo de página. Cabe destacar que la sincronización no se basa en turnos de acceso, sino en la programación precisa del programador para que los threads se ejecuten de forma correcta. En este caso tenemos de ejemplo que el ingreso de paginas esta demarcado por el tiempo de sleep de entre 1 y 2 milisegundos que presentan los threads y no por cada que una de estas paginas envejezca. Los threads comparten un elemento en común, el Buffer, que les da acceso a la ejecución de las operaciones. Es importante tener en cuenta que el éxito de la implementación depende de la correcta sincronización y acceso a los recursos compartidos por los threads.

Tabla Datos

TamMatriz	4x4	6x6
Mp=1	47	26
Mp=2	107	70
Mp=3	191	94

Graficas



¿Qué pasaría si las matrices fueran almacenadas siguiendo columna-major orden?

Si las matrices fueran almacenadas siguiendo column-major order en lugar de row-major order, esto podría afectar el rendimiento del programa en algunos casos debido a que el numero de paginas que se van a listar no tendría un orden uniforme como si lo tiene al ser en row-major-order. Por ejemplo, si el programa realiza muchas operaciones de lectura o escritura en las matrices, la lectura o escritura de elementos adyacentes podría requerir el acceso a diferentes bloques de memoria, lo que podría ser menos eficiente en términos de tiempo de ejecución. Sin embargo, en muchos casos, el impacto en el rendimiento sería insignificante, y la elección entre row-major y column-major order dependería de otros factores, como la facilidad de programación y la compatibilidad con otras bibliotecas y herramientas.

¿Cómo varía el número de fallas de páginas si el algoritmo estudiado es el de multiplicación de matrices?

El número de fallas de página puede variar significativamente entre el algoritmo de multiplicación de matrices y el de suma de matrices debido a la diferencia en la forma en que se acceden a los datos en memoria.

En el algoritmo de suma de matrices, se accede a los elementos de ambas matrices de forma secuencial y en nuestro caso siguiendo el row-major-order. Esto significa que los elementos que se acceden se encuentran generalmente cerca uno del otro en la memoria, lo que reduce la cantidad de fallas de página necesarias.

En cambio, en el algoritmo de multiplicación de matrices, los elementos de las matrices se acceden en patrones más complejos, lo que puede llevar a una mayor cantidad de fallas de página. Por ejemplo, se accede a cada elemento de la matriz A varias veces en el proceso de multiplicación, mientras que la matriz B se accede de forma no secuencial.

En general, se espera que el algoritmo de multiplicación de matrices tenga más fallas de página que el de suma de matrices. Sin embargo, la cantidad exacta de fallas de página dependerá de varios factores, como el tamaño de las matrices, la cantidad de memoria disponible y la forma en que se implemente el algoritmos. Además, debemos tener en cuenta que el algoritmo de suma de matrices tiene una mejor localidad temporal y espacial en comparación con la multiplicación.
El principio de localidad temporal como sabemos hace referencia a la tendencia de un programa de acceder repetidamente a la misma ubicación de memoria en un periodo corto de tiempo. En el algoritmo de suma, se accede a cada elemento de la matriz solo una vez lo que significa que hay menos posibilidades de tener que recuperar una pagina de memoria. Por otro lado la localidad espacial se refiere a la tendencia de acceder a ubicaciones de memoria cercanas. En el algoritmo de suma los elementos de la matriz se acceden de manera secuencial lo que produce que los elementos adyacentes estén mas cerca en la memoria y son mas propensos a estar en la misma pagina.

Conclusiones generales

En resumen el algoritmo de envejecimiento y reemplazo de página implementado en la opción 2 se basa en una estructura de Producer-Consumer, en la que dos threads, Thread1 y Thread2, comparten una misma clase Buffer para acceder a la tabla de marcos de página y actualizarla de forma sincronizada.

La tabla de marcos de página es una estructura que aloja información acerca de un número determinado de páginas, especificado por el tamaño del marco de página que el usuario proporciona. Cada entrada en la tabla contiene información sobre una página, como el número de página y un número que representa cuán recientemente fue referenciada esa página en la tabla de referencias (entre menor sea el número, menor tiempo desde la última referencia).

La clase Buffer se encarga de sincronizar el acceso a la tabla de marcos de página, permitiendo que solo un thread acceda a ella en un momento dado. En este caso, el Thread1 actúa como un Producer/Consumer, ya que es responsable de recorrer la tabla de referencias y actualizar en la tabla de marcos de página cuál fue la última página accedida. Si la tabla de marcos de página se llena, se produce un fallo de página, lo que implica que se debe hacer un swap entre la referencia a la página más antigua y la página que se quiere ingresar. Después de hacer una operación sobre la tabla de marcos de página, el Thread1 se duerme por 2 milisegundos antes de intentar leer la siguiente página referenciada en la tabla de referencias.

Por otro lado, el Thread2 también actúa como un Producer/Consumer, y es responsable de ejecutar el algoritmo de envejecimiento. Este algoritmo consiste en aumentar el tiempo de última referencia de cada página en la tabla de marcos de página. Después de actualizar este valor, el Thread2 se duerme por 1 milisegundo antes de volver a intentar ingresar a la tabla.

Es importante destacar que la sincronización no depende de los turnos de acceso, sino que depende exclusivamente del programador y de cómo se va a ejecutar el algoritmo. En este caso, no se meten páginas cada vez que se envejece, sino que se meten cada 1 o 2 milisegundos. La sincronización es crucial para garantizar que el algoritmo se ejecute correctamente, y en este caso, se utiliza la clase Buffer para lograrla.

Interpretación de Resultados

Basado en los resultados encontramos que uno de los elementos mas sensibles que influyen en el fallo de pagina es el numero de marcos disponibles y la cantidad de datos, esto debido a que si tenemos pocos marcos y muchas paginas debido al que el numero de datos es grande se van a tener muchos fallos de paginas al tener que recorrer todas estas paginas con tan pocos marcos disponibles. Esto a su vez también se ve representado en el tamaño de pagina ya que si el tamaño de pagina es pequeño van haber pocos datos por pagina y esto generara que si hay muchos datos, pocos marcos y poco espacio en pagina existan demasiadas paginas que deban ser referenciadas.
Por otro lado tenemos la manera en que sea recorrida ya que si el recorrido como en un columna major order no es secuencial, los datos quedaran repartidos y se tendrán que hacer varios llamados para recuperar los datos de una estructura.
Por último si el algoritmo cambia también el numero de fallas pues el funcionamiento de recorrido y ordenamiento cambiara afectando la manera en que las paginas serán llamadas para referenciar los datos necesarios para dar solución al algoritmo