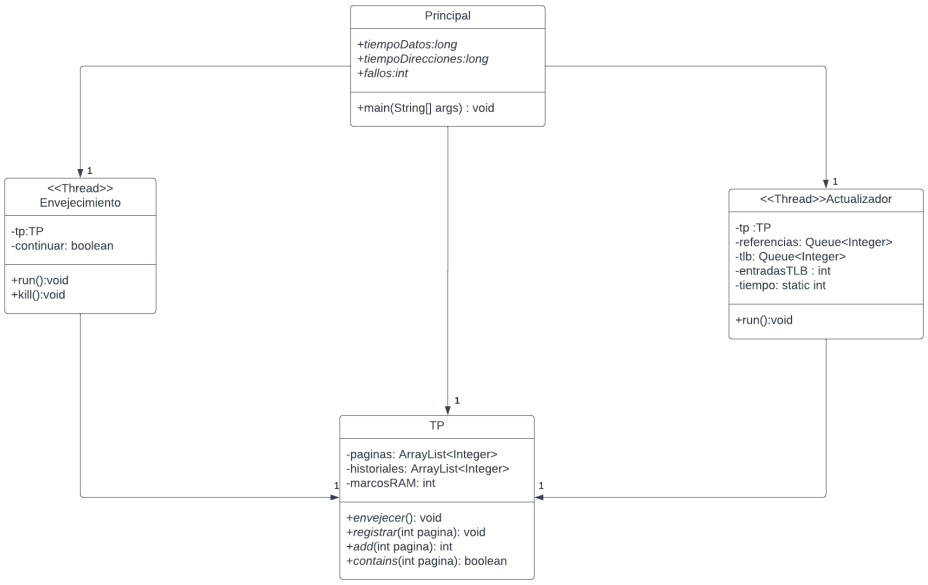
Infraestructura Computacional

Informe Caso 2

Laura Daniela Arias – 202020621
Josué Rivera – 201914138
Sebastián Andrés Ospino Salinas – 201913643

Diagrama de clases



Se implementaron 4 clases para simular el modelo de administración de memoria cuando se ejecuta un proceso: una clase *Principal* donde iniciar la simulación, una clase *Envejecimiento* para ejecutar el algoritmo de envejecimiento, una clase *Actualizador* para manejar las actualizaciones y los registros de la memoria, y una clase *TP* para representar la tabla de páginas.

Descripción de las clases

Principal
Como indica su nombre, esta es la clase principal, que contiene el método main(), y lleva a cabo las acciones necesarias para la simulación: pide al usuario los datos de entrada requeridos, realiza la lectura del archivo, crea la TP y la TLB, crea el *thread* de envejecimiento y el *thread* de actualización. Adicionalmente, tiene los contadores de tiempo de los datos y las direcciones, y el contador de fallos. Cuando finaliza la ejecución del programa, imprime en consola los datos de los tres contadores.

Envejecimiento
Esta clase—que extiende Thread—, se encarga de llevar a cabo en corrimiento de los 32 bits que indican las referencias hechas a cada página. Para lograr esto sin errores al acceder a los datos se implementa sincronización.

TP
La clase es la representación de la tabla de páginas requerida para la simulación. *TP* tiene como atributos dos *ArrayList<Integer>* que representan las 2 columnas que hacen parte de la tabla: el número de la página y los 32 bits del historial de referencias, respectivamente.

Actualizador
Esta clase—que extiende Thread—, se encarga de actualizar la TP y la TLB, y hacer los cálculos de los tiempos y los fallos de página.

Estructuras de datos

De las estructuras de datos ofrecidas por Java, en la simulación del sistema de paginación se hace uso exclusivo de *ArrayList<>*. Esta estructura se utiliza para guardar las referencias del archivo, representar la TLB, y para representar las dos columnas de la TP. En lo que refiere a las referencias, esta estructura permite iterar y con eso procesar ordenadamente todas las referencias de página. En la TLB, para el trabajo solo nos importa saber si una página se encuentra cargada en esa o no, y el orden en el que se encuentran las páginas en ella. Inicialmente, se pensaría en usar *Queue<>* para poder llevar a cabo el algoritmo FIFO del que hace uso la TLB, sin embargo, usar esta estructura implicaría no poder retirar de ella una página en específico cuando se elimine esta de la TP por motivos de espacio. Finalmente, tenemos dos *ArrayList<>* más en la clase *TP*, las cuales, como se mencionó anteriormente, representan 2 columnas de la TP: la columna de los números de página y los 32 bits de las referencias hechas a cada página. Al manejar las columnas por separado, podemos hacer consultas sobre el dato necesario fácilmente y, al almacenar la información sobre una misma página en un mismo índice, no perdemos la conexión entre datos. Solo es necesario hacer uso de métodos como *indexOf()* y *get()* para acceder el dato de una columna con respecto a la otra.

Esquema de sincronización

La sincronización presente en la arquitectura se centra en *TP*, que representa una estructura de datos compartida por los *thread*s del proyecto. Todos los métodos de la clase realizan algún tipo de modificación sobre los datos de la tabla, por lo que es necesario implementar sincronización para evitar que, en las acciones que llevan a cabo los *thread*s, estos "colisionen" al editar la TP y se genere un error.

Funcionamiento

Para terminar de explicar la arquitectura del programa, presentaremos un poco el detalle del funcionamiento del mismo.

```
public class Principal
{
    public static int fallos = 0;
    public static long tiempoDatos = 0;
    public static long tiempoDirecciones = 0;

    public static void main(String[] args) throws InterruptedException
    {
        try {
Scanner in = new Scanner(System.in);
            System.out.print("Ingrese el numero de entradas de la TLB: ");
            int entradasTLB = in.nextInt();
            System.out.print("Ingrese el numero de marcos de pagina en memoria RAM: ");
            int marcosRAM = in.nextInt();
            System.out.print("Ingrese el nombre del archivo con las referencias: ");
            String nombreArchivo = in.next();
            in.close();
```

```
            BufferedReader lector = new BufferedReader(new FileReader("./data/" + nombreArchivo));

            ArrayList<Integer> referencias = new ArrayList<>();

            String linea = lector.readLine();
            while (linea != null)
            {
                referencias.add(Integer.parseInt(linea));
                linea = lector.readLine();
            }
            lector.close();

            ArrayList<Integer> tlb = new ArrayList<>();
            TP tp = new TP(marcosRAM);

            Actualizador actualizador = new Actualizador(referencias, tp, tlb, entradasTLB);
            Envejecimiento envejecimiento = new Envejecimiento(tp);

            actualizador.start();
            envejecimiento.start();
```

Comenzamos pidiendo al usuario los datos de entradas de la TLB, marcos de página en RAM, y el nombre del archivo con las referencias. Luego, se cargan las referencias del archivo a un *ArrayList<>* y se crean las representaciones de la TLB y la TP (en la TP hacemos uso del dato de *marcosRAM* para alocar la memoria justa). Posteriormente creamos los dos *threads* y los corremos.

```
import java.util.ArrayList;

public class TP
{
    private ArrayList<Integer> paginas;
    private ArrayList<Integer> historiales;
    private int marcosRAM;

    public TP(int marcosRAM)
    {
        this.paginas = new ArrayList<Integer>(marcosRAM);
        this.historiales = new ArrayList<Integer>(marcosRAM);
        this.marcosRAM = marcosRAM;
    }
```

```
public class Actualizador extends Thread
{
    private TP tp;
    private ArrayList<Integer> referencias;
    private ArrayList<Integer> tlb;
    private int entradasTLB;
```

```
    public Actualizador(ArrayList<Integer> referencias, TP tp, ArrayList<Integer> tlb, int
entradasTLB) {
        this.tp = tp;
        this.referencias = referencias;
        this.tlb = tlb;
        this.entradasTLB = entradasTLB;
    }
```

```
public class Envejecimiento extends Thread
{
    private TP tp;
    private boolean continuar;

    public Envejecimiento(TP tp)
    {
        this.tp = tp;
        this.continuar = true;
    }
```

Dentro del método *run()* de *Actualizador* las cosas s desarrollan de la siguiente manera:

```
public void run() {
        for (Integer referencia: referencias)
        {
            if (!tlb.contains(referencia))
            {
                if (!tp.contains(referencia))
                {
                    Principal.fallos += 1;
                    int pagina = tp.add(referencia);

                    if (pagina >= 0 && tlb.contains(pagina))
                    {
                        tlb.remove(tlb.indexOf(pagina));
                    }

                    Principal.tiempoDirecciones += 60;
                    Principal.tiempoDatos += 10000000;
                }
                else
                {
                    Principal.tiempoDirecciones += 30;
                    Principal.tiempoDatos += 30;
                }

                if (tlb.size() == entradasTLB)
                {
                    tlb.remove(0);
                }
```

```
                tlb.add(referencia);
            }
            else
            {
                Principal.tiempoDirecciones += 2;
            }

            tp.registrar(referencia);

            try {
                Thread.sleep(2);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

En resumidas cuentas, cuando la referencia no se encuentra en la TLB ni en la TP (es decir, ocurre un fallo de página), se adiciona el fallo al contador, se agrega la referencia a la TP y, si fue necesario eliminar una página de la TP, se elimina también de la TLB. Luego se agregan los tiempos de la operación y se agrega la nueva página referenciada a la TLB, sin olvidar llevar a cabo el algoritmo FIFO si la TLB está completa. Si la referencia no estaba en la TLB, pero sí en la TP, se adicionan los tiempos del proceso y se realiza la isma operación de insertar la nueva referencia en la TLB. Si, por el contrario, la referencia se encontraba ya en la TLB, solo es necesario registrar la referencia dentro de los 32 bits de la tabla de páginas (este proceso se lleva a cabo en todos los casos) y sumar el tiempo de la operación.

Los siguientes son los métodos de la clase *TP*:

```
    public synchronized boolean contains(int pagina)
    {
        return paginas.contains(pagina);
    }

    public synchronized int add(int pagina)
    {
        int paginaTLB = -1;

        if (paginas.size() == marcosRAM)
        {
            ArrayList<Integer> copia = new ArrayList<Integer>(historiales);
            copia.removeIf(n -> n<0);
            copia.sort(null);
            int indice = historiales.indexOf(copia.get(0));
            paginaTLB = paginas.get(indice);
            paginas.remove(indice);
            historiales.remove(indice);
        }
```

```
        paginas.add(pagina);
        historiales.add(0);

        return paginaTLB;
    }

    public synchronized void envejecer()
    {
        for (Integer historial: historiales)
        {
            historial >>= 1;
        }
    }

    public synchronized void registrar(int pagina)
    {
        int historial = historiales.get(paginas.indexOf(pagina));
        historial |= ~(2^31-1);
    }
}
```

Estos métodos son aquellos que presentan la sincronización al tratarse de datos compartidos por los *threads*. Nos gustaría hacer especial énfasis en *add()* y *registrar()*.
En *add()* hacemos el proceso de agregar una página a la TP. Para esto, si la TP no puede albergar más elementos, se evalúa la página a eliminar según el algoritmo de envejecimiento. Como el dato está representado como un *int*, si el bit más significativo fuera 1 el entero sería negativo y, por ende, falsamente el menor. Para manejar esto creamos una copia de *hisroriales* que solo toma en cuenta los datos de valor positivo y sobre ese sacamos el dato más pequeño. Adicionalmente, para poderle indicar luego a la TLB si tiene que eliminar una página de su lista se usa la variable *paginaTLB* que toma el valor de -1 si no es necesario eliminar una página o el valor del número de página si debe de eliminarla.
Por su parte, *envejecer()* es el método utilizado para agregar el bit de referencia de una página. Para esto se hace un *o* lógico del dato con el complemento del número 2^{31}-1. Se hizo de esta manera para evitar problemas con el signo del entero y sugerir más directamente los bits requeridos.

Datos

A partir del archivo de prueba dado y de las instrucciones de evaluar los casos:

Número de marcos en RAM: 8, 16, 32

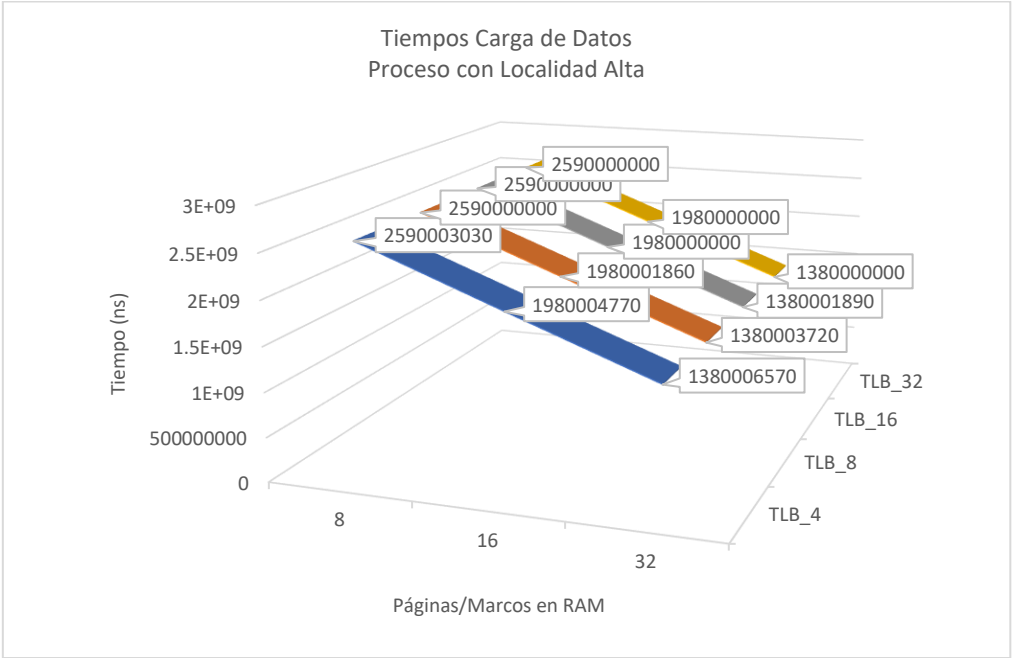
Entradas de la TLB: 8, 16, 32, 64

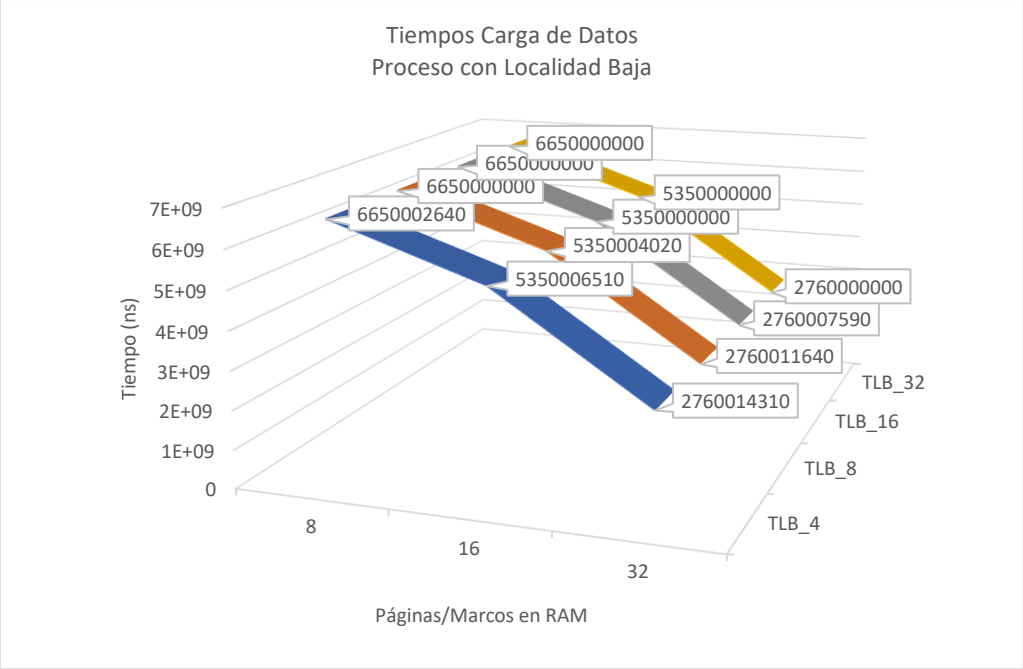
Pudimos recompilar y organizar la siguiente información a partir de nuestras ejecuciones y pruebas.

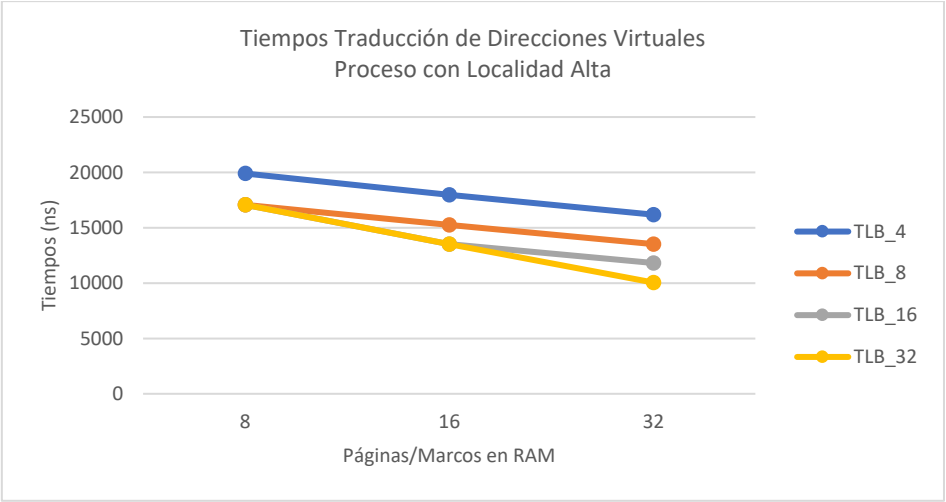
Tabla de resultados

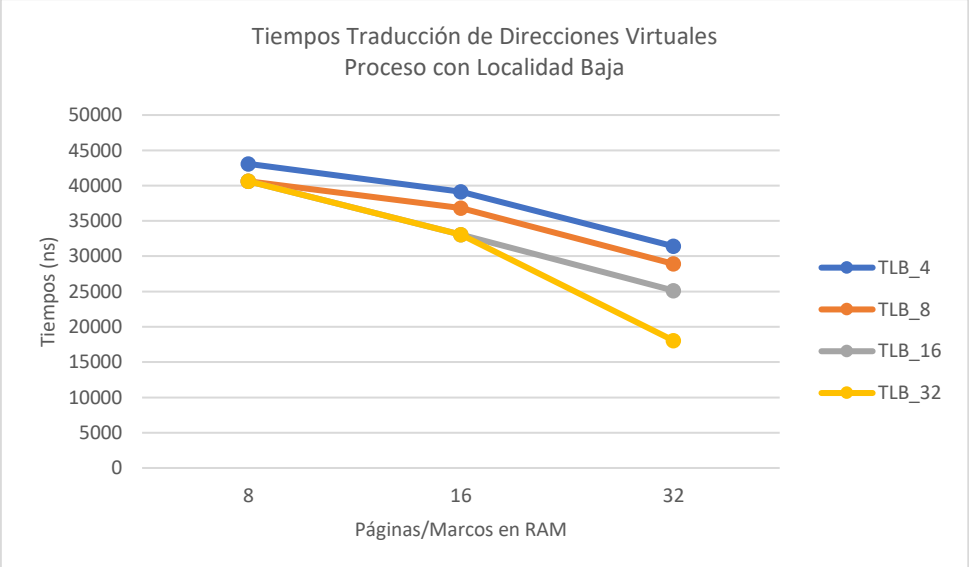
TLB	PÁGINAS/MARCOS EN RAM	LOCALIDAD ALTA		LOCALIDAD BAJA	
		TIEMPO CARGA DE DATOS (NS)	TIEMPO TRADUCCIÓN DE DIRECCIONE VIRTUALES (NS)	TIEMPO CARGA DE DATOS (NS)	TIEMPO TRADUCCIÓN DE DIRECCIONE VIRTUALES (NS)
TLB_4	8	2590003030	19898	6650002640	43082
	16	1980004770	17984	5350006510	39154
	32	1380006570	16184	2760014310	31412
TLB_8	8	2590000000	17070	6650000000	40618
	16	1980001860	15268	5350004020	36830
	32	1380003720	13524	2760011640	28920
TLB_16	8	2590000000	17070	6650000000	40618
	16	1980000000	13532	5350000000	33078
	32	1380001890	11816	2760007590	25140
TLB_32	8	2590000000	17070	6650000000	40618
	16	1980000000	13532	5350000000	33078
	32	1380000000	10052	2760000000	18056

Gráficas de datos




Tiempos Carga de Datos
Proceso con Localidad Baja
Tiempo (ns)
7E+09
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
6650000000
6650000000
6650000000
6650002640
5350000000
5350000000
5350004020
5350006510
2760000000
2760007590
2760011640
2760014310
TLB_32
TLB_16
TLB_8
TLB_4
8
16
32
Páginas/Marcos en RAM


Tiempos Traducción de Direcciones Virtuales
Proceso con Localidad Alta
Tiempos (ns)
25000
20000
15000
10000
5000
0
8
16
32
Páginas/Marcos en RAM
TLB_4
TLB_8
TLB_16
TLB_32


Tiempos Traducción de Direcciones Virtuales
Proceso con Localidad Baja
Tiempos (ns)
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
8
16
32
Páginas/Marcos en RAM
TLB_4
TLB_8
TLB_16
TLB_32

Conclusión

En los datos evidenciados en las gráficas, se observa que entre más localidad tengan las referencias del proceso, existen muchos menos fallos de páginas, lo que se traduce en menores tiempos. En específico, el promedio de tiempo para la simulación con referencias de baja localidad fue aproximadamente de 5 segundos, mientras que con alta localidad fue aproximadamente de 2 segundos, lo que hace que la ejecución de esta última sea 2.5 veces más rápida con respecto a la primera. Otro factor que también influyó en las diferencias de tiempo fue el número de marcos en RAM. Es de esperarse que entre más números de marcos de página en RAM (espacio de trabajo) se le asignen a un proceso durante su ejecución se obtengan mejores tiempos, puesto que entre más páginas pueda tener el proceso en memoria física, es menos probable que ocurran fallos de páginas. Esto se observa en que el porcentaje de mejora promedio, con respecto al tiempo original, del tiempo de simulación es en promedio del 27% cuando se agregan 8 marcos de páginas más. Las mejoras de tiempos también se ven influidas por el número de entradas en la TLB. En nuestro caso, se observa que las mejoras son insignificantes en comparación con los 2 factores anteriormente mencionados, una posible explicación para este fenómeno puede ser que los fallos de páginas toman varios órdenes de magnitud más en solucionarse que el tener que acceder a RAM por un dato que un principio no se encontró en la TLB. Por último, queríamos recalcar como nuestros datos evidencian la relación entre la TLB y la TP. Al ser la TLB un “subconjunto” de la TP, mientras el número de marcos de página sea menor al número de entradas de la TLB, los tiempos de ejecución se mantienen iguales, señalando que lo que ocurra con y en la TLB depende fuertemente de las “herramientas” que disponga la TP. Creemos que todos estos resultados tienen sentido y