

Universidad de los Andes

Facultad de Ingeniería
Departamento de Ingeniería de Sistemas
Infraestructura Computacional
Periodo 2022-20



Caso 2 Documentación

Juan Diego Yepes Parra: 202022391
Paula Alejandra Molina Ruiz: 201631596
Nicolás Angarita Moreno: 201812581

13 de octubre de 2022
Bogotá D.C.

Tabla de contenidos

1	Problema	2
1.1	Enunciado	2
1.2	Especificación	2
2	Solución	2
2.1	Modelo de dominio	2
2.1.1	Descripción de las estructuras de datos usadas	3
2.1.2	Por qué la sincronización	4
2.2	Esquema de actualización de estructuras de datos	4
2.2.1	Flujo de ejecución	4
2.2.2	Aging.java	6
2.2.3	ReferenceUpdate.java	6
3	Resultados	7
3.1	Interpretación de resultados	10
3.2	Captura de ejecución	10
4	Instalación y repositorio	11

1 Problema

A continuación se detalla el problema a resolver en este caso.

1.1 Enunciado

La memoria RAM es un recurso limitado que debe administrarse con cuidado para garantizar el avance en la ejecución de los procesos creados. En este contexto surge el concepto de memoria virtual, el cual ofrece varias ventajas: independencia de direcciones físicas, posibilidad de compartir memoria y de correr programas más grandes que la memoria física que se les asigna.

Queremos comprender un poco mejor cómo varía el tiempo de ejecución de un proceso de acuerdo con las características de un sistema de memoria virtual, en particular, TLB y espacio asignado en RAM. Su tarea en este caso es escribir un programa en Java que calcule el tiempo total que invierte el sistema para resolver direcciones virtuales y para cargar las páginas necesarias en memoria RAM, incluyendo el tiempo la resolución de fallas de página para que un proceso se ejecute. Para la resolución de fallas de página usaremos el algoritmo de envejecimiento; Tanenbaum explica este algoritmo en su libro Sistemas Operativos Modernos, capítulo 3 – sección 3.4.7 “Simulación de LRU en software” (disponible en versión electrónica en la biblioteca, edición 3, año 2009).

1.2 Especificación

A continuación se presenta una especificación (informal) del programa a desarrollar:

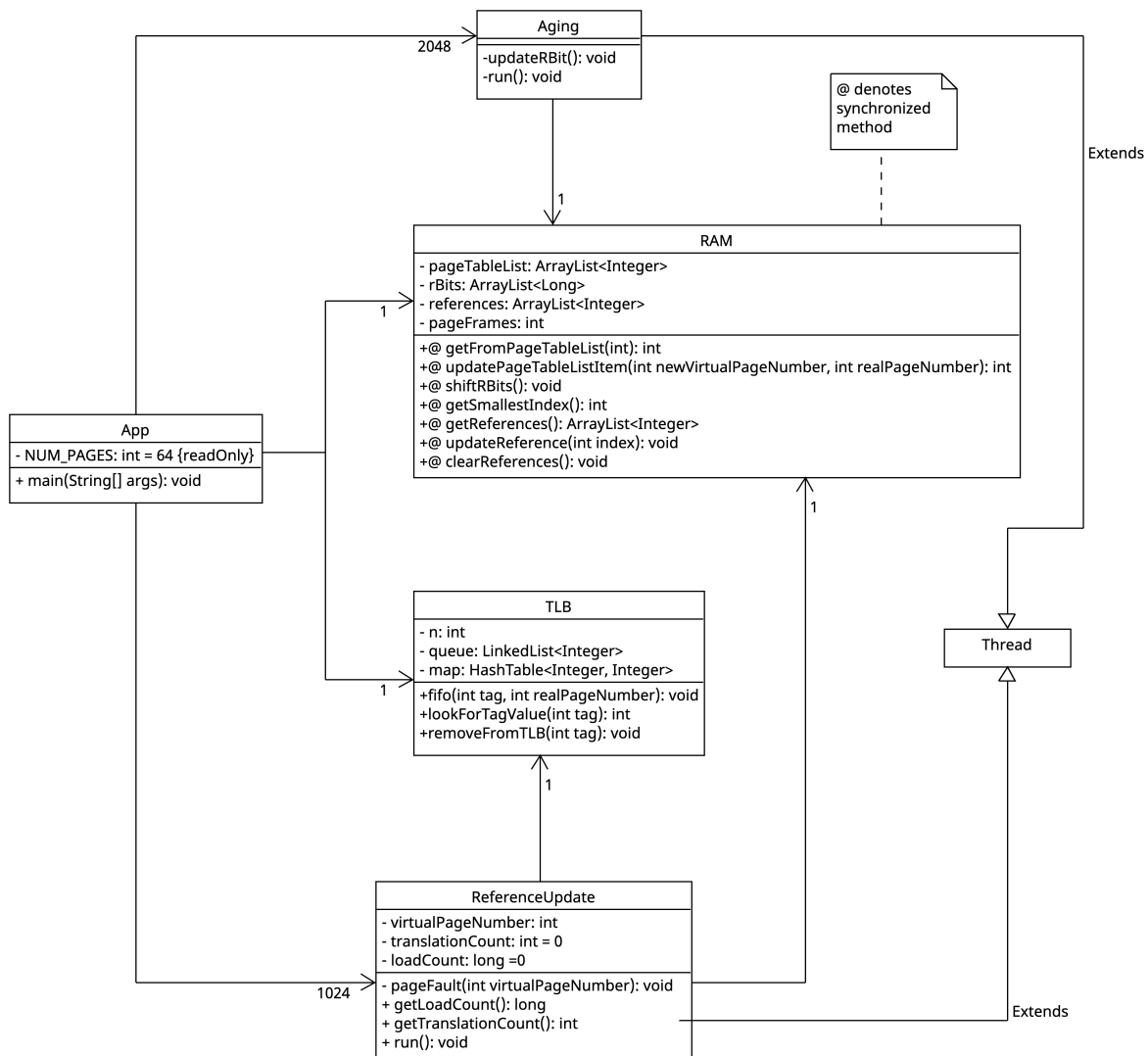
I/O	Nombre	Tipo	Descripción
I	<i>nTLB</i>	int	Número de entradas de la TLB
I	<i>nPF</i>	int	Número de marcos de página en memoria RAM
O	<i>translation</i>	int	Tiempo gastado por el proceso en traducción

2 Solución

A partir de lo anterior, proponemos la siguiente solución al problema.

2.1 Modelo de dominio

Para el desarrollo del caso se plantea el siguiente diagrama de dominio:



El flujo entre clases y cómo se intercambian información está detallado en la sección 2.2. Por otro lado, se utilizó el caracter @ para denominar que un método es synchronized. Este diagrama y el código están escritos en inglés.

2.1.1 Descripción de las estructuras de datos usadas

- Clase RAM

- `pageTableList`: Esta lista representa la tabla de páginas. Cada índice de la lista es el número de página virtual y el valor es el número de página real. Si dicho número de página virtual no está presente en la RAM, el valor es -1

- `rBits`: Esta lista tiene el `rBit` de cada número de página real en RAM. Funciona con el algoritmo de envejecimiento.
- `references`: Esta lista tiene la referencia de cada página real en un instante determinado. Funciona con el algoritmo de envejecimiento.
- Clase `TLB`
 - `queue`: Cola para implementar el algoritmo FIFO. Funciona como un `LinkedList`
 - `map`: Mapa de parejas llave, valor que mapea el TLB. La llave es el número de página virtual y el valor es el número de página real.

2.1.2 Por qué la sincronización

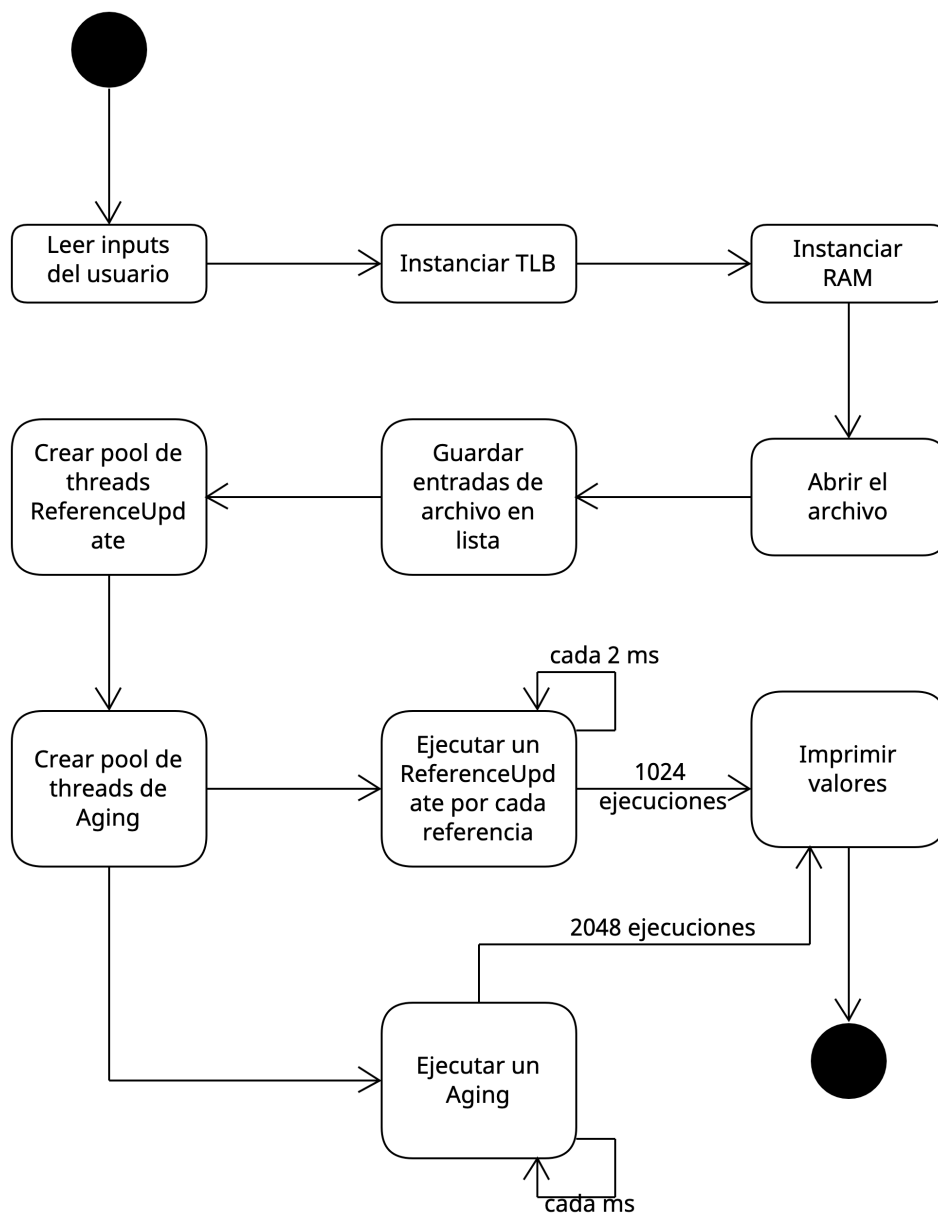
Como se muestra en el modelo de dominio, todos los métodos que interactúen con `pageTableList`, `rBits` y `references` son sincronizados. Esto es importante porque estas listas son compartidas entre los threads que ejecutan el algoritmo de envejecimiento `Aging.java` y los que ejecutan la carga de cada referencia de página `ReferenceUpdate.java`. De no haber esta sincronización podrían haber lecturas sucias en las que no se tome el bit más pequeño, la TLB guarde las referencias erróneas, etc.

2.2 Esquema de actualización de estructuras de datos

A continuación se detalla, mediante el flujo de ejecución, cómo se actualiza cada estructura de datos

2.2.1 Flujo de ejecución

A manera de un diagrama de flujo (con sintaxis libre sin seguir un lenguaje de modelado determinado) mostraré el flujo del programa el cual será explicado a continuación:



El programa empieza recibiendo cada input, que serían el número de entradas de la TLB, el número de marcos de página en RAM y el nombre del archivo que contiene las referencias. Luego, se agrega a una lista cada referencia dentro del archivo. Posteriormente, se instancian cada uno de las clases necesarias de acuerdo con el flujo de ejecución, para los threads se hace una especie de pool de threads (una lista con muchas instancias). Luego, en un ciclo que recorre la lista del pool de threads se inicializan así:

```

1  for (ReferenceUpdate r:
2      referenceUpdates) {
3      agings.remove(0).start();
4      Thread.sleep(1);
5      agings.remove(0).start();
6      r.start();
7      r.join();
8      loadCount += r.getLoadCount();
9      translationCount += r.getTranslationCount();
10     Thread.sleep(1);
11 }

```

Con esto se asegura que se ejecutan cada milisegundo las de envejecimiento `Aging.java` y cada dos milisegundos las de traducción y carga `ReferenceUpdate.java`.

2.2.2 Aging.java

El algoritmo de envejecimiento es bastante simple. En primer lugar, se actualizan los `rBit` de cada referencia, es decir, por cada elemento dentro de la lista `references` que tenga un 1, se le pone un 1 al bit 32 del número de página real. Posteriormente, se corren todos los `rBits` a la derecha y finalmente se resetean las `references` para la siguiente iteración del algoritmo.

```

1  @Override
2  public void run() {
3      updateRBit();
4      ram.shiftRBits();
5      ram.clearReferences();
6  }

```

2.2.3 ReferenceUpdate.java

El algoritmo de actualización de referencias tiene que tener varios procesos, para facilidad de explicación se muestra el código fuente:

```

1  @Override
2  public void run() {
3      int physPageNumber = tlb.lookForTagValue(virtualPageNumber);
4      if (physPageNumber != -1 && physPageNumber != -2){ //tlb hit
5          translationCount += 2;
6          loadCount += 30;
7      } else { // looks on the page table
8          physPageNumber = ram.getFromPageTableList(virtualPageNumber);
9          if (physPageNumber == -1){ // page fault
10             pageFault(virtualPageNumber);
11             physPageNumber = ram.getFromPageTableList(virtualPageNumber);
12             tlb.fifo(virtualPageNumber, physPageNumber);
13         }

```

```
14         else { // its on RAM
15             translationCount += 30;
16             tlb.fifo(virtualPageNumber,physPageNumber);
17         }
18     }
19 }
```

1. Se busca en la TLB la referencia.

- (a) Si está en la TLB (es decir que el valor no es -1 ni -2), fue un hit en la TLB y se aumentan los valores correspondientemente
- (b) Si no está en la TLB tiene que entrar a ver la tabla de páginas

2. Se busca en la tabla de páginas

- (a) Si está en RAM se suman los tiempos correspondientes y se agrega al caché TLB
- (b) Si no está en RAM hay un fallo de página, el cual se resuelve leyendo dos veces la tabla de páginas. Se escoge un índice a sacar (gracias al algoritmo de envejecimiento) y se actualizan la tabla de páginas y la TLB si hay dependencias.

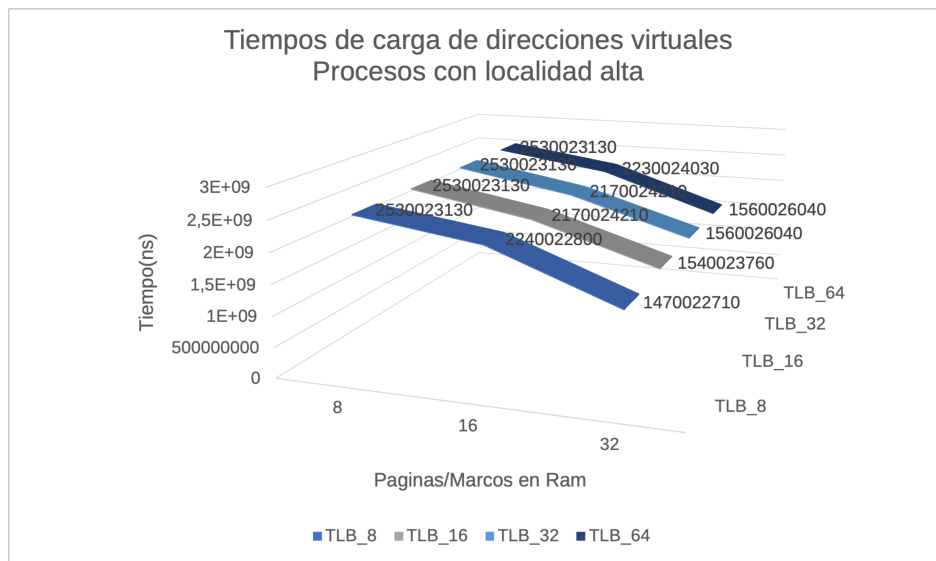
3 Resultados

TLB size	Page frames in RAM	Translation Time	Loading Time
8	8	16722	2530023130
16	8	16722	2530023130
32	8	16722	2530023130
64	8	16722	2530023130
8	16	16160	2240022800
16	16	14634	2170024210
32	16	14634	2170024210
64	16	14982	2230024030
8	32	13934	1470022710
16	32	13164	1540023760
32	32	11096	1560026040
64	32	11096	1560026040

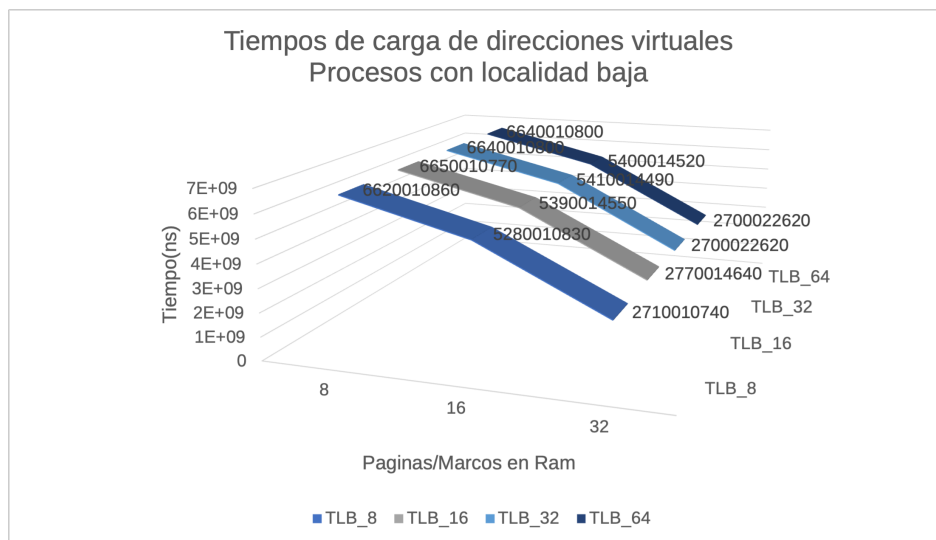
Tabla 1: Localidad alta

TLB size	Page frames in RAM	Translation Time	Loading Time
8	8	40444	6620010860
16	8	40618	6650010770
32	8	40560	6640010800
64	8	40560	6640010800
8	16	36452	5280010830
16	16	33310	5390014550
32	16	33426	5410014490
64	16	33368	5400014520
8	32	28826	2710010740
16	32	25366	2770014640
32	32	17708	2700022620
64	32	17708	2700022620

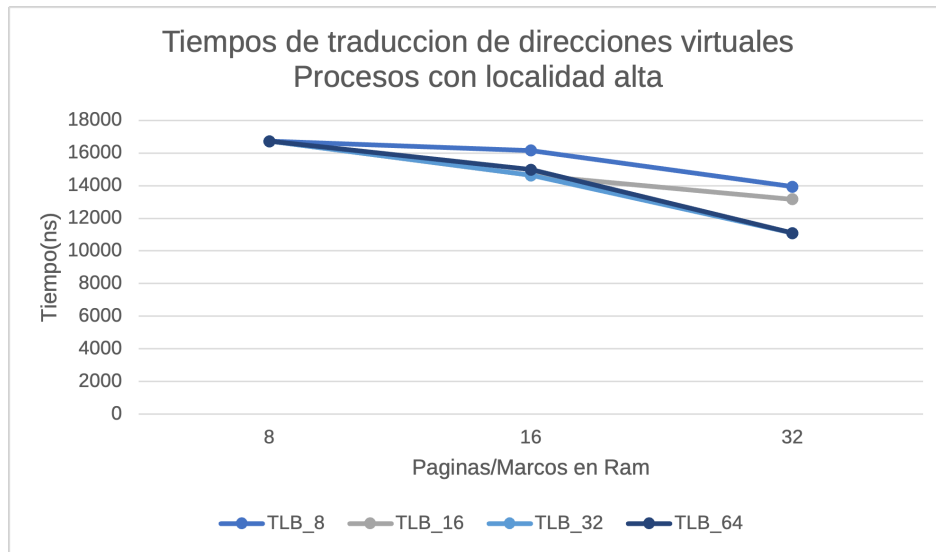
Tabla 2: Localidad baja



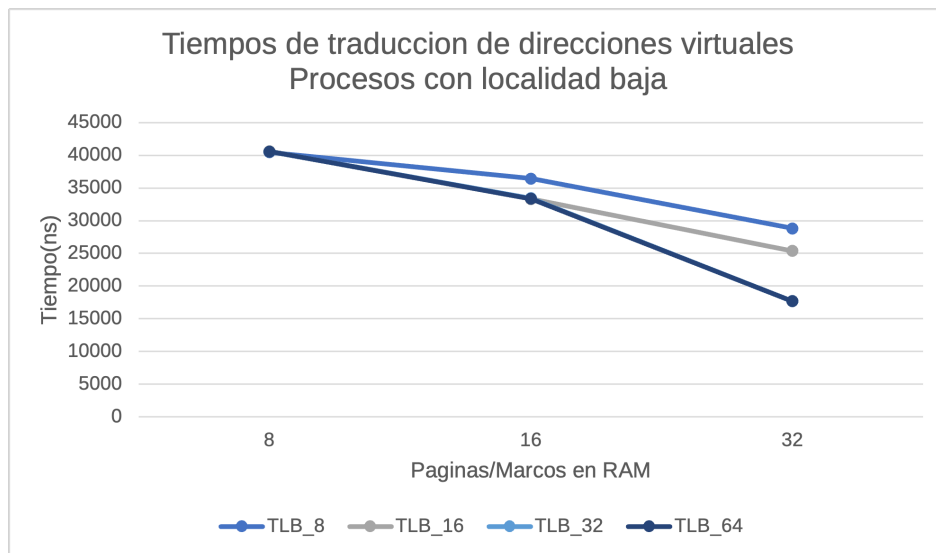
Gráfica 1: Tiempos de carga de datos en localidad alta



Gráfica 2: Tiempos de carga de datos en localidad baja



Gráfica 3: Tiempos de traducción de datos en localidad alta



Gráfica 4: Tiempos de traducción de datos en localidad baja

3.1 Interpretación de resultados

De acuerdo a los datos obtenidos podemos obtener las siguientes conclusiones:

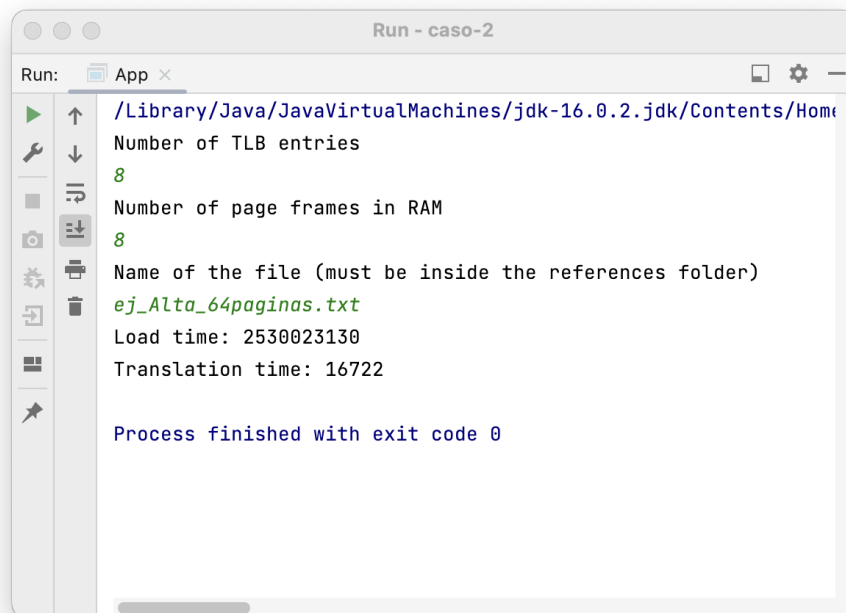
1. Los datos tienen sentido y son acordes con lo que debería pasar
 - (a) A mayor número de marcos de página, menor debe ser el tiempo de traducción y

de carga puesto que se van a dar menos fallos de página porque hay más páginas directamente en RAM

- (b) Las TLB con capacidades más grandes tienen tiempos menores de traducción, ya que este caché puede abarcar más páginas y por ende no hay que ir hasta la tabla de páginas. Sin embargo en tiempo de carga no es diferente porque igual los datos están en RAM.
- 2. Existen algunas discordancias con los datos propuestos, esto se puede deber a que el algoritmo puede estar calculando diferente algunos tiempos.
- 3. Sin embargo, realizando las pruebas con los datos pequeños (que inician por `test_`, podemos ver que los fallos de página son exactamente los propuestos, luego en cuanto hits y fallos de página nuestro programa ejecuta correctamente.

3.2 Captura de ejecución

A continuación se muestra una captura del programa ejecutándose para demostrar su correcto funcionamiento:



```
Run - caso-2
Run: App x
/Library/Java/JavaVirtualMachines/jdk-16.0.2.jdk/Contents/Home
Number of TLB entries
8
Number of page frames in RAM
8
Name of the file (must be inside the references folder)
ej_Alta_64paginas.txt
Load time: 2530023130
Translation time: 16722

Process finished with exit code 0
```

4 Instalación y repositorio

Para acceder al repositorio del código fuente diríjase al siguiente [link](#).

Para correr el programa, clone el repositorio y desde IntelliJ abra el archivo `./src/App.java` que es el que contiene el método `main()`. Ejecutarlo dando click en Run. También es posible copiar todos los archivos `.java` en un proyecto nuevo de Eclipse, y ejecutarlo.