**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

**PROYECTO FINAL DE SISTEMAS OPERATIVOS**

**“EVALUACIÓN DE UNA CACHÉ DE METADATOS PARA UN SISTEMA DE ARCHIVOS”**

**Dra. CRISTINA ABAD R.**

**INTEGRANTES**

**ANGELY OYOLA**

**RODRIGO CASTRO**

**JHON BARREIRO**

**LENÍN TENECELA**

**2015 – 2T**

**INTRODUCCIÓN**

El sistema de archivos es el conjunto de métodos y estructuras de datos que un sistema operativo utiliza para administrar los archivos de un disco o partición y así tener una organización sobre ellos.

En la actualidad la cantidad de datos que se manipulan a través de archivos, ya sea creando, accediendo, modificando o guardando, es muy grande; por dicho motivo los sistemas operativos alojan su sistema de archivos en almacenamiento secundario y lo estructuran de tal forma que el tiempo de respuesta en el acceso a los datos sea óptimo, la información esté protegida, se note un buen rendimiento, haya soporte a requerimientos de los usuarios, entre otros.

Dicha estructura está dividida en capas que en conjunto resuelven los problemas de diseño y organización de los archivos, ya que se debe utilizar la forma más natural a la hora de mostrar la estructura de archivos al usuario y lo más importante es utilizar algoritmos y estructuras de datos que optimicen el tiempo de respuesta al momento de acceder a los datos.

Una de las capas en cuestión es la de “Sistemas de Archivos Lógicos” que principalmente es en donde se encuentran los metadatos con los “paths” de los archivos y el esquema depende del sistema operativo que se esté utilizando. Por ejemplo, en un SO. Unix como GNU/Linux se sigue el siguiente formato: “/wikipedia/commons/thumb/e/e0/Icono\_aviso\_borrar.png/50px-Icono\_aviso\_borrar.png”, mientras que en Windows: “C:\Users\Aoyola\CacheMemory\workload.txt”.

Indiferente del formato de las rutas de los archivos, los sistemas operativos utilizan memoria caché para reducir el tiempo de respuesta en el acceso a los archivos sobre todo a la hora de acceder a memoria, a esto se añade una correcta elección de estructuras de datos para el alojamiento del directorio general y el uso de algoritmos de desalojo de caché óptimos  ya que esto afecta directamente el rendimiento, eficiencia y fiabilidad del sistema de archivos en general.

Por todo lo descrito, se ha solicitado implementar una caché, desarrollar y analizar 4 algoritmos de desalojo de caché: LRU, LRU-k, CLOCK y ÓPTIMO con un dataset descargado de “<http://www.wikibench.eu/wiki/2007­10/>”, en el que cada fila representa la ruta de acceso a los archivos y así poder conocer el rendimiento de cada uno de los algoritmos variando parámetros, tales como el tamaño de la caché.

**KEYWORDS**

Metadatos, Sistema de Archivos, Sistema de Archivos Lógicos, Algoritmos de Desalojo, Caché.

**METODOLOGÍA**

La siguiente sección describe las estructuras de datos utilizadas en el proyecto. Aquellas fueron un punto esencial para facilitar la inserción, búsqueda y eliminación de datos y así optimizar los tiempos finales de respuesta de los diferentes algoritmos. Dichas estructuras pueden ser divididas en 2 grupos:

1. Generales y utilizadas por todos los algoritmos de desalojo de la caché.

**Nodelist**: Elemento básico para las demás estructuras. Va ligado netamente con la estructura List.

**List**: Elemento fundamental para el uso de la estructura Queue, permite la organización de elementos así como la modificación de los mismos.

**Queue:** Elemento basado en la estructura List. Creado con el fin de tener el comportamiento de un cola de datos.

**Item:** Estructura fundamental para la organización de los metadatos. Conformado por un nodelist, índice, la jerarquía del archivo y un entero denominado bit, que será útil para el algoritmo CLOCK.

**Cubeta:** Estructura en la que se van a encontrar los ítems. Se aloja en la tabla hash y ayuda a que ésta sea una tabla Hash abierta y así evitar colisiones.

**Hashtable:** Es aquella estructura esencial del proyecto que nos permite la manipulación de datos con tiempos de respuesta óptimos. Se basa en un conjunto de cubetas para evitar las muy conocidas colisiones y en un entero que es el resultado de aplicar la funcuín hash.

**Memory Cache:** Es la memoria caché del proyecto conformada principalmente por una tabla Hash, ítems, números de hits, misses, misses inevitables, tamaño de la caché, entre otras.

1. Específicas propias de cada algoritmo.

Para la ejecución de los algoritmos se utilizaron principalmente 3 funciones. B1) Test creados para cada uno de ellos y sirven de vinculación directa con lo ingresado por consola. Aquí se evalúa el archivo que contiene los paths y el tipo de algoritmo a utilizar. Aquí se da el formato de presentación de misses, hits, missrate, hitRate y warm cache.

B2) La función real en donde nos basamos en estructuras que nos permitan realizar lo planificado en un tiempo mínimo.

B3) Y la función que permite la inserción de datos a la caché.

Es importante recalcar que para cada uno de los 4 algoritmos se realizaron las 3 funciones mencionadas anteriormente.

A continuación se describen los algoritmos.

**LRU:** Este algoritmo consiste en dar un vistazo a los datos que ya han sido utilizados y así desalojar aquellos que tienen menos frecuencia de uso tomando en cuenta el tamaño de la caché. Para su funcionamiento nos basamos en el uso de la estructura QUEUE y de la memoria caché.

**LRUK:** Es una variante del LRU y tiene el fin de desalojar de la caché verificando la frecuencia de acceso de la k-ésima vez de uso. Se utilizan las mismas estrcutras antes descritas más una variable entera que nos permitirá aplicar los algoritmos sin dificultad.

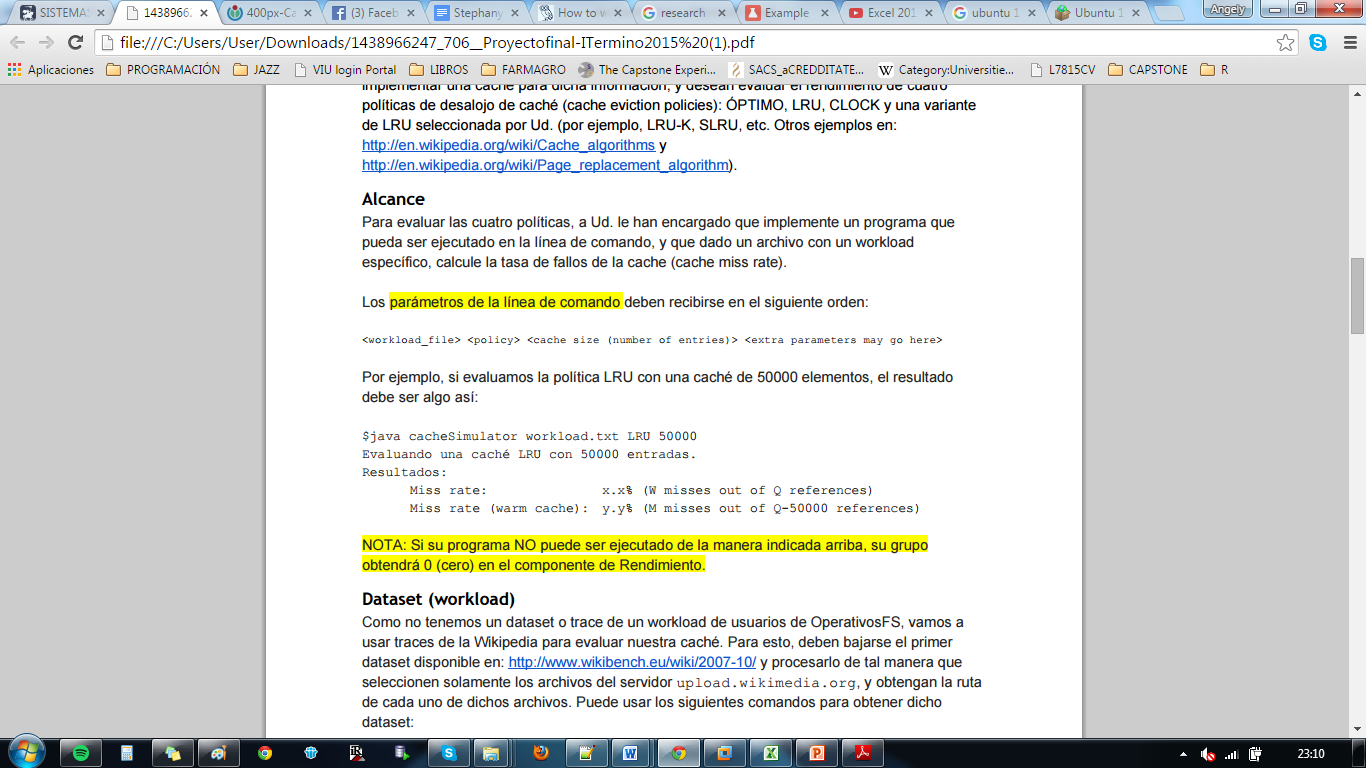
**CLOCK:** Se asemeja al LRU en casi todo a excepción que este utiliza un bit el cual otorga una segunda oportunidad para que un archivo no referenciado no sea desalojado de la caché. Para su funcionamiento nos basamos en el uso de la estructura QUEUE y de la memoria caché.

**ÓPTIMO:** En algoritmos de desalojo, el óptimo es aquel que todos desean por su correcto performance y tiempo de respuesta. La desventaja es que en casos de la vida real no se puede predecir el futuro; es decir, no se puede pensar qué solicitud vendrá, entonces como la base de este algoritmo es mirar hacia adelante, es difícil de implementar. A parte de utilizar la memoria caché y otras estrucutras ya mencionadas, el Óptimo utiliza un Heap que permite ordenar los elementos de manera eficaz.

**RESULTADOS**

Se ejecutaron por línea de comando los 4 algoritmos escogidos: LRU, Clock, LRU-K y óptimo. Esto se realizó en una máquina virtual con sistema operativo GNU/Linux Ubuntu 14.

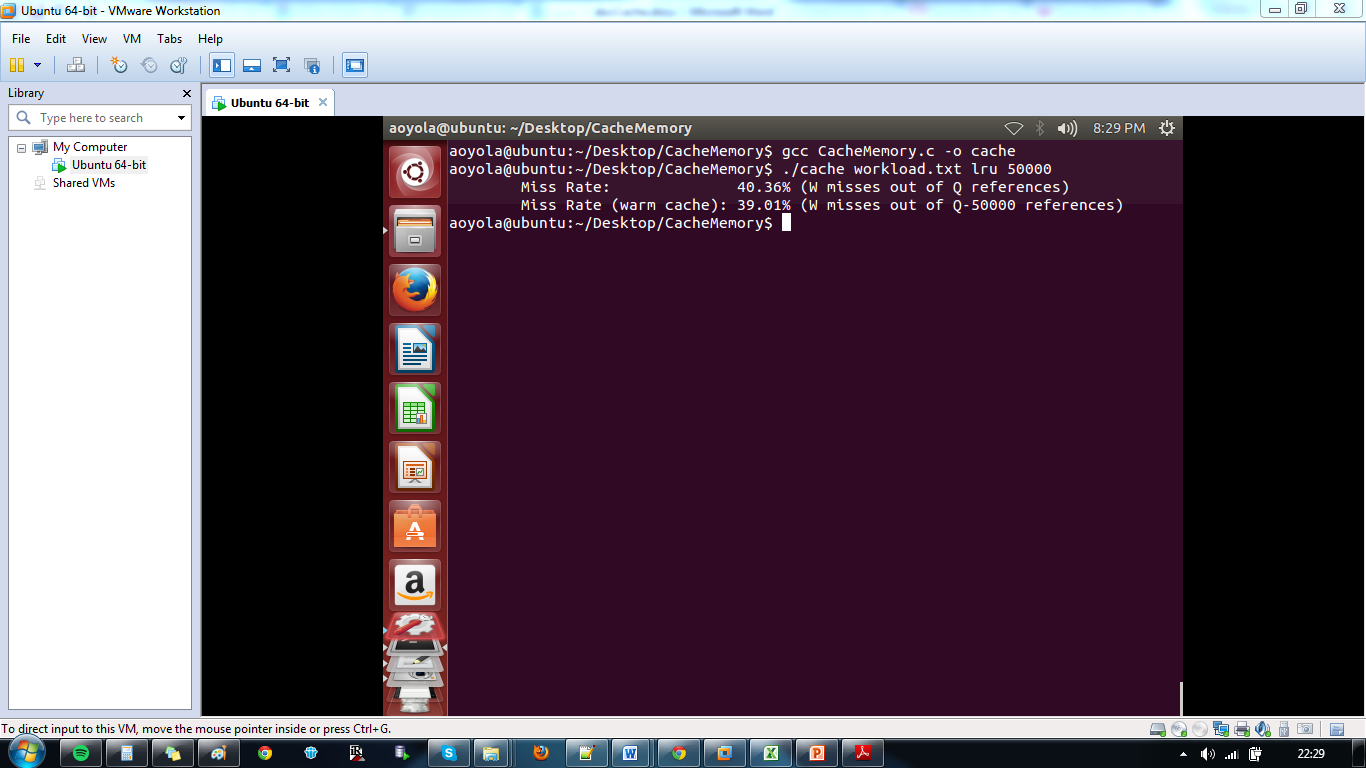
El formato seguido en la ejecución de los algoritmos fue el siguiente:



A continuación los resultados por cada algoritmo.

**LRU**

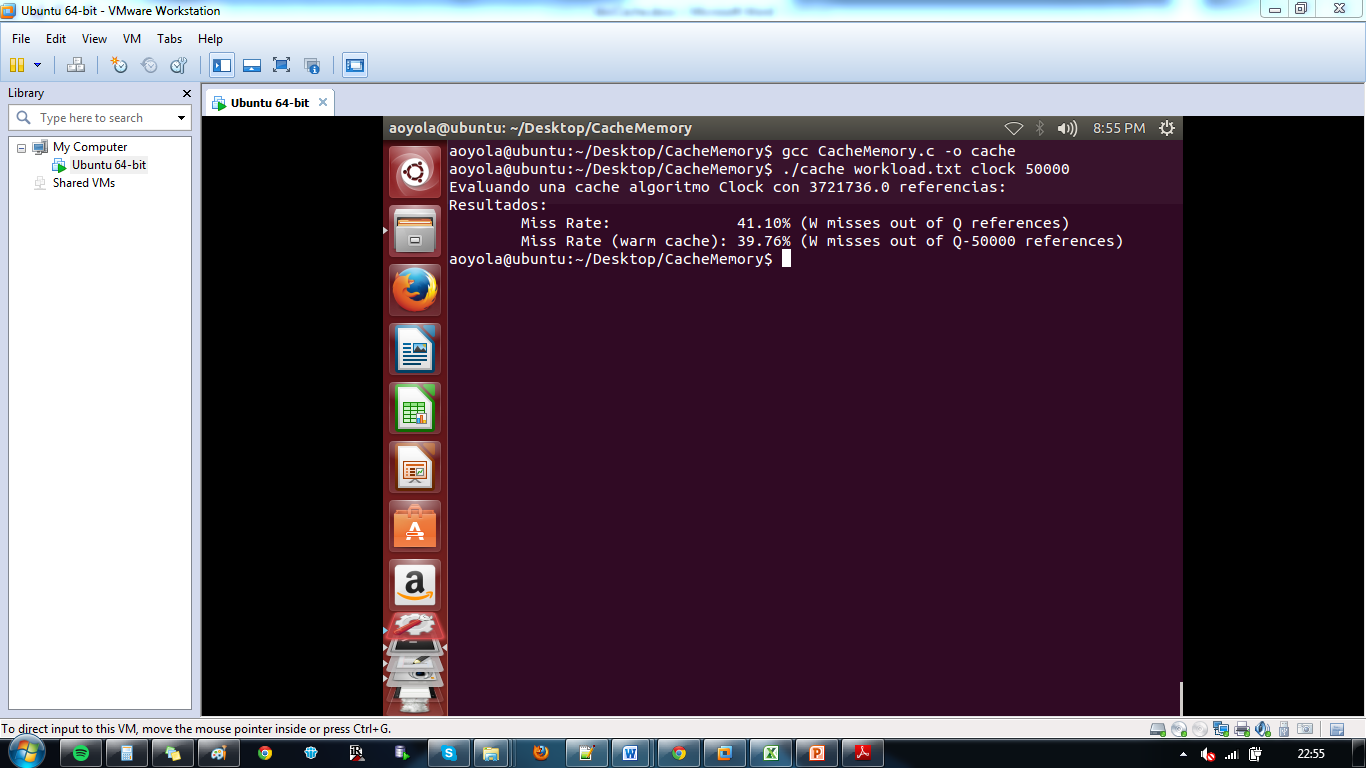
1. **Ejecución utilizando el algoritmo LRU con tamaño de caché = 50.000 frames.**



**Observación**: Se ejecutó el algoritmo LRU con 3721736 datos del archivo “workload.txt” y con un tamaño de caché de 50000 frames. Se obtuvo una tasa de misses del 40.36%, es decir que más de la mitad fueron hits lo que hace al algoritmo no haya sido tan mala elección. Otro aspecto a considerar es la warm caché, en lo que nos detalla que un 39.01% de misses son inevitables si se trabaja con dicho tamaño de caché.

**CLOCK**

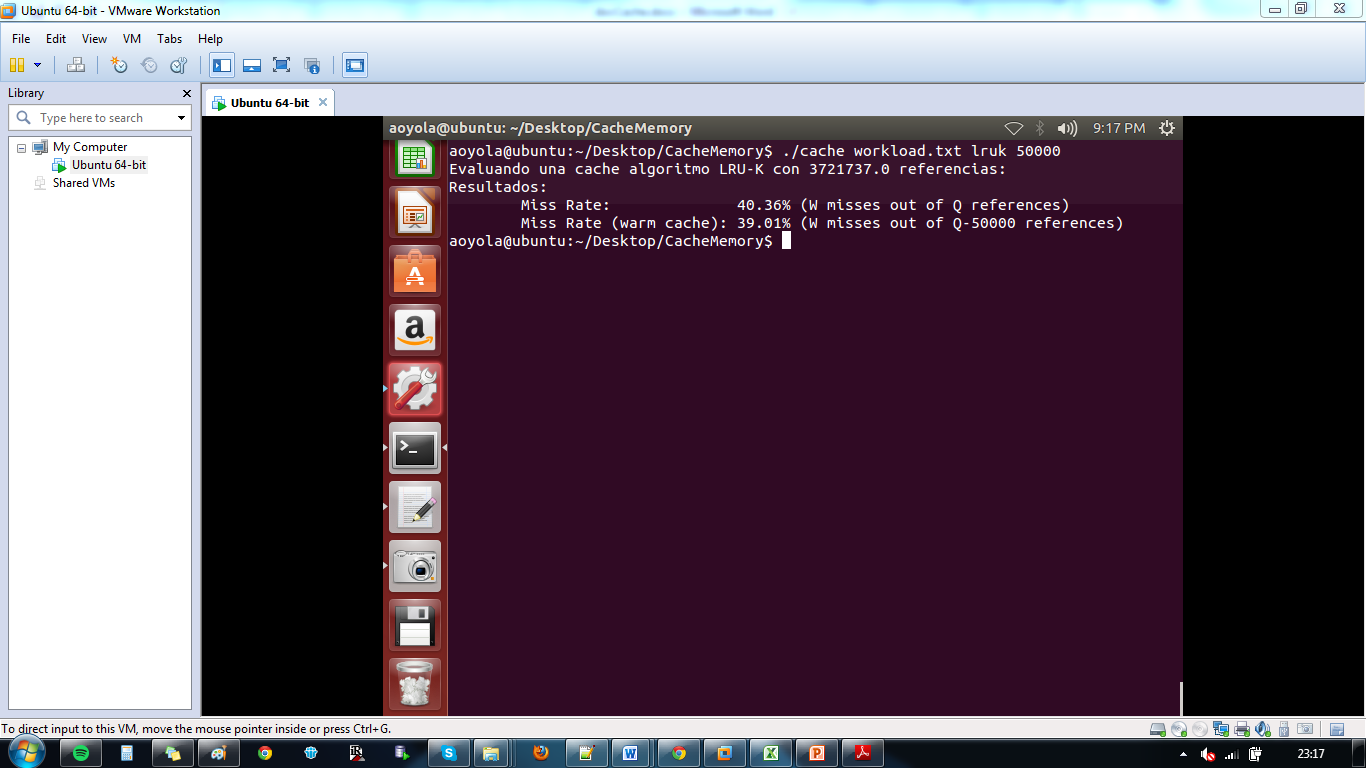
1. **Ejecución utilizando el algoritmo CLOCK con tamaño de caché = 50.000 frames.**



**Observación**: Se ejecutó el algoritmo CLOCK con 3721736 datos del archivo “workload.txt” y con un tamaño de caché de 50000 frames. Se obtuvo una tasa de misses del 41.10%, es decir que el LRU lo supera en 0.74 %. Otro aspecto a considerar es la warm caché, en lo que nos detalla que un 39.76% de misses son inevitables si se trabaja con dicho tamaño de caché.

**LRU-K**

1. **Ejecución utilizando el algoritmo LRUK con tamaño de caché = 50.000 frames.**



**Observación**: Se ejecutó el algoritmo LRUK con 3721736 datos del archivo “workload.txt” y con un tamaño de caché de 50000 frames. Se obtuvo una tasa de misses del 40.36%, es decir que el LRUk es tan efectivo como lo es el LRU trabajando con 50000 frames ya que poseen la misma tasa de misses. Dejando al último al CLOCK.

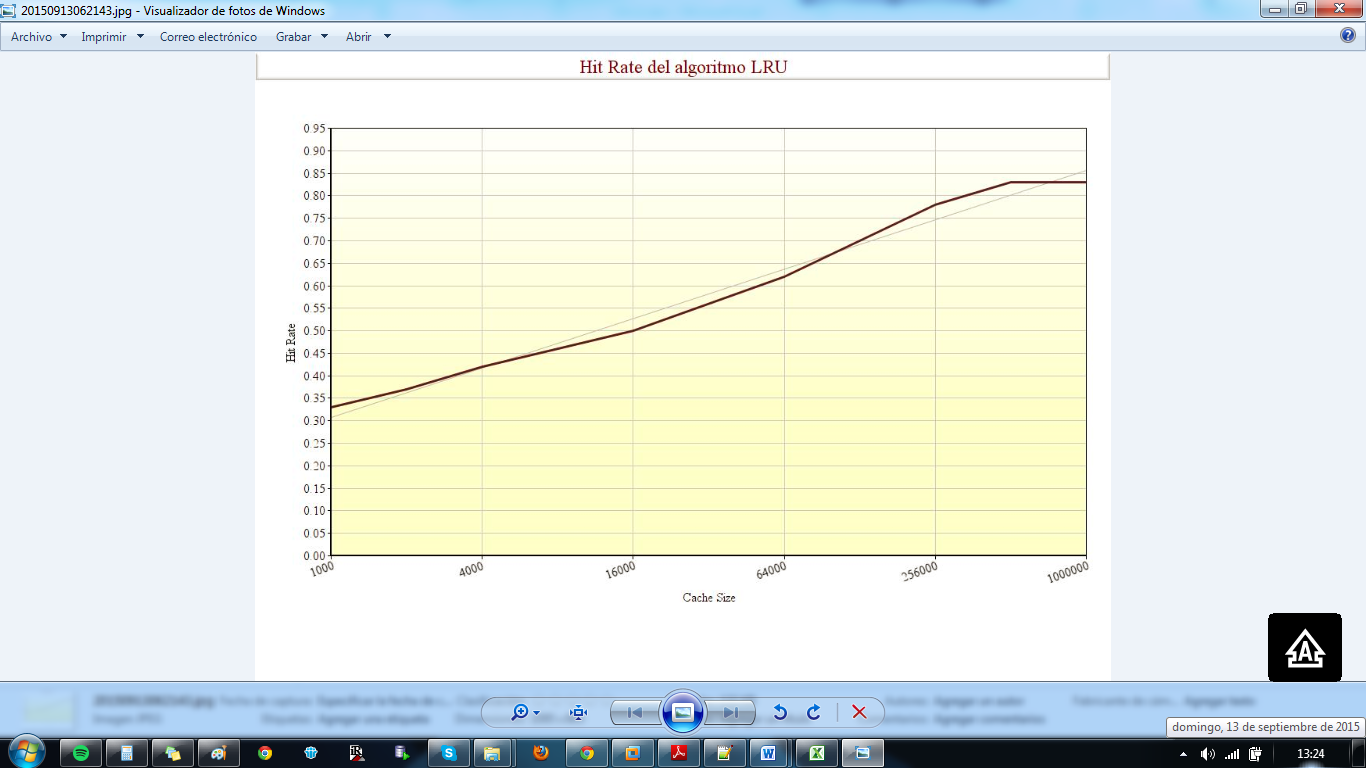
**OPTIMO**

1. **Ejecución utilizando el algoritmo ÓPTIMO con tamaño de caché = 50.000 frames.**

Para poder comparar la eficiencia de los algoritmos se optó por medir los hitRate de diferentes tamaños de caché. Los valores variables de caché fueron de: 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K y 1 M.

A continuación se muestran las gráficas de HitRate obtenidas por cada algoritmo.

**LRU**



|  |  |  |  |
| --- | --- | --- | --- |
| **HITS** | **HIT RATE** | **MISSES** | **MISS RATE** |
| 1216465 | 0,3269 | 2505271 | 0,6731 |
| 1385069 | 0,3722 | 2336667 | 0,6278 |
| 1545490 | 0,4153 | 2176246 | 0,5847 |
| 1704111 | 0,4579 | 2017625 | 0,5421 |
| 1870357 | 0,5025 | 1851379 | 0,4975 |
| 2066479 | 0,5552 | 1655257 | 0,4448 |
| 2314074 | 0,6218 | 1407662 | 0,3782 |
| 2611670 | 0,7017 | 1110066 | 0,2983 |
| 2904120 | 0,7803 | 817616 | 0,2197 |
| 3081684 | 0,8280 | 640052 | 0,1720 |
| 3093419 | 0,8312 | 628317 | 0,1688 |

**Observación**: Con este gráfico podemos notar que el límite de HitRate obtenido con el LRU es del 85% ya que a partir de que la caché tome valores mayores o iguales a 1M el valor de hitRate se mantiene en 0.83. Esto evidencia que a mayor tamaño de la caché mayo es el performance, pero en el caso del LRU no se puede superar al 85%.

**CLOCK**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **HITS** | **HIT RATE** | **MISSES** | **MISS RATE** |
| 1176091 | 0,3160 | 2545645 | 0,6840 |
| 1354061 | 0,3638 | 2367675 | 0,6362 |
| 1516313 | 0,4074 | 2205423 | 0,5926 |
| 1677011 | 0,4506 | 2044725 | 0,5494 |
| 1846853 | 0,4962 | 1874883 | 0,5038 |
| 2042198 | 0,5487 | 1679538 | 0,4513 |
| 2284269 | 0,6138 | 1437467 | 0,3862 |
| 2574331 | 0,6917 | 1147405 | 0,3083 |
| 2866204 | 0,7701 | 855532 | 0,2299 |
| 3055793 | 0,8211 | 665943 | 0,1789 |
| 3093419 | 0,8312 | 628317 | 0,1688 |

**Observación**: Con este gráfico podemos notar que el algoritmo CLOCK es menos eficiente que el LRU antes descrito ya que con valores entre 1K y 1M de caché tiene un promedio de HitRate del 57.37% mientras que el LRU tiene uno del 58.12%.

**LRUK**



|  |  |  |  |
| --- | --- | --- | --- |
| **HITS** | **HIT RATE** | **MISSES** | **MISS RATE** |
| 1215708 | 0,3267 | 2506029 | 0,6733 |
| 1384716 | 0,3721 | 2337021 | 0,6279 |
| 1545330 | 0,4152 | 2176407 | 0,5848 |
| 1704018 | 0,4579 | 2017719 | 0,5421 |
| 1870312 | 0,5025 | 1851425 | 0,4975 |
| 2066446 | 0,5552 | 1655291 | 0,4448 |
| 2314058 | 0,6218 | 1407679 | 0,3782 |
| 2611661 | 0,7017 | 1110076 | 0,2983 |
| 2904115 | 0,7803 | 817622 | 0,2197 |
| 3081685 | 0,8280 | 640052 | 0,1720 |
| 3093419 | 0,8312 | 628318 | 0,1688 |

**Observación**: Con este gráfico podemos notar que el algoritmos LRUK es tan eficiente como lo es el LRU antes descrito ya que con valores entre 1K y 1M de caché se obtuvo un promedio de HitRate del 58.11% casi igual al 58.12% obtenido con el LRU.

**ÒPTIMO**

**CONTRASTE ENTRE LOS ALGORITMOS**

**CONCLUSIONES**

* Con una memoria caché que fluctúa entre 1K y 1M no se encontraron grandes diferencias entre los algoritmos LRU y CLOCK, mientras que con el LRUK sí.
* Con las pruebas respectivas a todos los algoritmos se dedujo que el límite a la hora de hablar de HitRate es del 85%; es decir que no existe ningún algoritmo que por más cantidad de memoría caché utilizada nos brinde un hitRate del 100%.