**Escuela Superior Politécnica del Litoral**

# “*Impulsando la sociedad del conocimiento"*

****

# Sistemas Operativos

**Proyecto final:**

**Evaluación de una caché de metadatos para sistemas de archivos**

**Integrantes:**

* **Ronny Morán**
* **Gustavo Londa**
* **Fabián Sayay**
* **Carlos Ramírez**

**Paralelo: 1**

**2015 - 2016**

**TABLA DE CONTENIDO**

**Capítulos Páginas**

|  |  |  |
| --- | --- | --- |
| **1** | **Introducción** | 3 |
| 2 | Objetivo | 3 |
| **3** | **Metodología** | 3 |
| 4 | Resultados | 5 |
| 5 | Conclusiones | 7 |

**1.0 Introducción:**

Una política de reemplazo en caché es una forma de decidir qué elemento desalojar, cuando la caché está llena. Los sistemas de archivos almacenan los metadatos asociados a un archivo de manera independiente de los datos o contenido del archivo. Estos metadatos incluyen el nombre del archivo, su longitud, y su ubicación en la jerarquía de archivos (ej.: /home/user/cabad/varios.txt). Dichos metadatos son almacenados en disco, en archivos con una estructura propietaria o en una base de datos ligera. Sin embargo, consultar esta base de datos cada vez que un usuario quiere abrir un archivo resulta costoso debido a lo lento que es leer del disco duro.

Por esta razón, los sistemas de archivos modernos, utilizan cachés en memoria que contienen información parcial de los metadatos. Cuando un usuario desea abrir un archivo, primero se consulta los metadatos en la caché, y solamente si no se encuentra la información en la misma (cache miss), entonces se consulta los metadatos en disco.

Este documento está dividido en las siguientes secciones:

2.0 Objetivos: En el cual encontramos los objetivos del Proyecto.

3.0 Metodología: En esta sección encontraremos el IDE y lenguaje utilizado para la implementación del Proyecto, y la estructura de datos utilizada.

4.0 Resultados: En esta sección se muestra el rendimiento de cada política implementada.

Finalmente las conclusiones.

**2.0 Objetivo:**

Implementar y evaluar una caché de metadatos para sistemas de archivos utilizando deferentes politicas de reemplazo (LRU, Óptimo, Clock, LFU) para cierto workload (trace de Wikipedia).

**3.0 Metodología:**

Para la implementación del Proyecto se utilizó el entorno de desarrollo integrado Ninja IDE para la programación en python.

Se implementaron las políticas de reemplazo:

LRU, ÓTIMO, CLOCK, LFU

Como workload se utilizaron traces de Wikipedia y se le estableció un cierto formato por parte de la profesora.

Para ejecutar el script del proyecto por línea de comando, establecemos como formato el siguiente:

<workload\_file> <policy> <cache size (number of entries)>

Ejemplo (evaluando una cache con 50000 entradas):

$python simulador-cache.py workload.txt LRU 50000

**Descripciones de Políticas de reemplazo implementadas.**

**LRU**

Se utilizaron listas para la implementación de este algoritmo en python, en el cual se verificaba si un url del workload está en caché, si no estaba y la cache aún no llegaba al límite (tamaño), se insertaba en caché (miss), cuándo la caché está llena, y el url a insertar, no se encuentra en la cache se procede a reemplazar por el menos reciente usado, el cual se llevaba el orden a travéz de otra lista para obtener el índice del “mas\_antiguo” y poder realizar el reemplazo por el nuevo url.

**Algoritmo Óptimo**

Este algoritmo tiene como finalidad retirar la página que vaya a ser referenciada más tarde. Como se puede deducir, para esto el sistema operativo debería ver en cuánto tiempo será usada cada página en memoria y elegir la que está más distante. El problema de este algoritmo es que necesita conocimiento del futuro, por lo que es imposible su implementación de manera optima. Como solo es un algoritmo teórico, en nuestra implementación (no óptima) utilizamos listas para poder recorrer las páginas que están disponibles en caché y asi poder remplazar la que se utilizara en el plazo más largo.

**CLOCK**

En el algoritmo clock , se usó un arreglo de dos dimensiones el cual se dividió de la siguiente manera: el arreglo estaba compuesto de arreglos que tiene un tamaño de 3, en el primer índice se almacena el puntero, en el segundo la palabra que se va a usar y en el tercero el bit se segunda oportunidad como por ejemplo:

[[Puntero, palabra, bit], [Puntero, palabra, bit] ………..].

El arreglo principal tendrá el tamaño de la caché, para la elaboración del algoritmo se dividió en 2 parte, la primera será para ingresar y comparando los datos hasta que se llene el arreglo, la segunda parte es cuando el arreglo está lleno se procederá a comparar que si esa palabra se encuentra ingresada, si lo está se pone el bit de segunda oportunidad igual a 1 y si no está se procede a reemplazar según el puntero.

**LFU**

Se utilizó un método de conteo para lo cual se implementó un diccionario de datos asociado a la frecuencia que las páginas han sido referenciadas, esto facilita la selección del elemento de menor Contador para ser reemplazado por la nueva página que será insertada en caché.

**3.0 Resultados:**

La figura1 nos muestra el algoritmo LRU (% Mis rate vs tamaño de caché)

El tiempo de ejecución de esta politica de reemplazo es mayor al aumentar el tamaño de la caché.

**FIGURA1**

La figura2 nos muestra el algoritmo Óptimo (% Mis rate vs tamaño de cache)

Se utilizaron listas para la implementacion de este algoritmo

**FIGURA2**

La fIgura3 nos muestra el algoritmo Clock (% Mis rate vs tamaño de caché)

El tiempo de ejecución de esta política de reemplazo es mayor al aumentar el tamaño de la caché.

**FIGURA3**

La fIgura4 nos muestra el algoritmo LFU (% Mis rate vs tamaño de cache)

El tiempo de ejecución de esta política de reemplazo es menor al aumentar el tamaño de la caché.

El tiempo de ejecución de esta política de reemplazo es menor al aumentar el tamaño de la caché.

**FIGURA4**

Todos los algoritmos utilizaron el mismo workload de wikipedia.

**5.0 Conclusión:**

* El tiempo de ejecución del algoritmo para cierta política de reemplazo depende del tamaño de la caché y del workload utilizado, además de las caracteristicas del computador (Ej: procesador).
* Al aumentar progresivamente el tamaño de cache en los 4 algoritmos, podemos observar que le porcentaje de miss rate no presenta mayor variación.

**LINK DE GITHUB DEL PROYECTO:**

**https://github.com/SOFinal/CacheAlgoritmo**