UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio 3 de Organización de Computadores

Integrantes:	Cristian Espinoza
Curso:	Organización de Computadores
Sección	L1
Profesor:	Daniel Wladdimiro
Ayudante:	Sebastian Pinto-Agüero

19 de Noviembre de 2017

Tabla de contenidos

Índice de figuras

1. Introducción

En el presente documento se le ha pedido a los alumnos de la asignatura Organización de computadores que presenten el trabajo realizado en el laboratorio numero tres, el cual al realizarlo conlleva el aprendizaje de los contenidos en dos niveles, de manera practica y teórica simultáneamente.

Como lo confirma la teoría en cátedra, en la organización de computadores, se ha aprendido que uno de los factores importantes que inciden en el rendimiento de un programa es la cantidad de accesos a memoria que debe realizar el procesador. En este laboratorio se estudia el acceso, que realiza la memoria caché (o las memorias caché), siendo estas, memorias más pequeña que la memoria principal de un computador. Estas memorias resultan ser limitadas, por lo tanto no pueden contener todos los datos existentes en la memoria principal, por lo que estos deben ser ingresados por bloques a una dirección determinada de la caché, y en caso de estar ocupada esa dirección, se utilizan ciertas políticas de reemplazo para, reemplazar el contenido anterior por el que está ingresando a la caché.

La solución del presente laboratorio se abarca con la herramienta de división en sub-problemas, con el fin de poder abarcar en su totalidad cada una de sus dificultades. Este es capaz de leer un archivos de entrada, el cual contiene las direcciones de palabras que se consultarán a la caché, en el momento que la caché se llena, se aplican técnicas de reemplazo para poder almacenar los datos consultados. Cabe destacar que el alcance que se presenta es el cumplimiento del desarrollo completo del laboratorio.

1.1. Motivación

El laboratorio propuesto por los docentes de la asignatura organización de computadores, permitirá desarrollar los conocimientos entregados en la cátedra respecto a la aplicación de la memoria caché y sus respectivas técnicas de reemplazo, para así mejorar el rendimiento de los alumnos dentro de la asignatura.

1.2. Objetivo General

Aprender, aclarar y profundizar el concepto de la memoria caché, desarrollado en lenguaje de programación C, ocupando el paradigma de programación imperativo.

1.3. Objetivos Específicos

Los objetivos específicos del programa son implementar lo siguiente:

- Realizar un archivo con los datos alojados en la caché posterior a la ultima consulta realizada.
- Generar un archivo en el cual se pueda visualizar las estadísticas del caché, considerando la cantidad de *Hit* , *Miss* y la tasa de cada uno de ellos.

1.4. Datos

- Un archivo que contiene las direcciones de memoria que se consultaran a la memoria caché.

1.5. Herramientas

Las herramientas utilizadas para la realización de este laboratorio son:

- *Sublime text 3*. (C)
- Gcc version 6.2.0.
- *Sharlatex*.

1.6. Estructura del informe

Por ultimo, en el presente informe se observa un marco teórico necesario para el entendimiento de todo el desarrollo del programa, además se cuenta con un desarrollo que explica cómo se realizo, las herramientas y técnicas utilizadas para el desarrollo de este.

2. Marco teórico

A continuación, se mostrará en forma clara los conceptos que se deben manejar para una adecuada lectura del informe.

2.1. Memoria caché

Una memoria caché es una memoria en la que se almacena una serie de datos para su rápido acceso. Guzmán et al. (2017). Es un tipo de memoria volátil, pero de una gran velocidad. Su finalidad es almacenar una serie de instrucciones y datos a los que el procesador accede continuamente, con la finalidad de que estos accesos sean instantáneos. LG (2016)

Esta memoria, puede estar agrupada en vías, donde cada vía cuenta con bloques de memoria, y cada bloque contiene cierta cantidad de palabras, que en este caso una palabra es un dato de 32 bits.

2.2. Configuración de caché

Una memoria caché tiene distintas configuraciones, y estas son las siguientes:

2.2.1. Mapeo directo

Estructura en la cual cada ubicación en memoria puede ser mapeada a sólo una localización en caché. Guzmán et al. (2017).

2.2.2. *Full* asociativo

Una estructura de cache en la cual un bloque puede ser guardado en cualquier localización en el caché. Guzmán et al. (2017)

2.2.3. N-vías asociativo o Conjunto-asociativo

Es idéntica a la configuración de mapeo directo, en el cual utilizamos una dirección para mapear un bloque a una determinada posición dentro de la caché, con la diferencia que en lugar de mapear un único bloque en caché, se hace a un conjunto de bloques. Todos los conjuntos en caché tienen el mismo tamaño.

2.3. *Hit* y *Miss*

- *Hit*: Una petición de un dato de cache que puede ser respondida del caché ya que el dato está presente. Guzmán et al. (2017)
- *Miss*: Una petición de un dato de cache que no puede ser respondida del caché ya que el dato no está presente. Guzmán et al. (2017)

2.4. Políticas de reemplazo

Cuando se tiene una caché con una configuración n-vías asociativas, y una de las vías posee todos sus bloques ocupados, se requiere una política de reemplazo que indique cual bloque debería ser reemplazado para dar lugar al nuevo dato.

2.4.1. MRU *(Most Recently Used)*

Un esquema de reemplazo en el cual el bloque reemplazado es el que ha sido usado más recientemente. Guzmán et al. (2017)

2.4.2. LRU *(Least Recently Used)*

Un esquema de reemplazo en el cual el bloque reemplazado es el que no ha sido usado por el mayor tiempo. Cada posición tiene un contador de cuantas veces se ha usado. Cuando ingresa un elemento nuevo el computador se resetea. En cada referencia el contador se incrementa en 1.Cuando un elemento se reutiliza su contador también se resetea. Se elimina el que tiene el número más alto. Guzmán et al. (2017)

2.4.3. FIFO *(First In First Out)*

Lo que significa primero en entrar será el primero en salir. Por lo tanto se irán reemplazando las líneas de la memoria cache que lleven más tiempo por las nuevas. Guzmán et al. (2017)

3. Desarrollo

3.1. Estructuras

- **Bloque**: Esta estructura se encarga de representar un bloque de la memoria caché y almacena las palabras que se colocan en él, las cuales se representa por un arreglo de tipo *int*. Además, se cuenta con una variable de tipo *int LRU*, la cual sirve al momento de tener que utilizar la política de reemplazo LRU. En la figura 1, se aprecia la estructura.

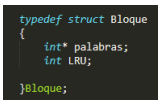

Figura 1: Bloque

- **Vías**: Esta estructura se encarga de representar las vías de la memoria caché. Además, almacena dos variables de tipo *int MRU* y *contador*, las cuales sirven al momento de utilizar las políticas de reemplazo *FIFO* y *MRU*. Finalmente se cuenta con un puntero de tipo *Bloque*, el cual contiene todo los bloques de las *Vías*. En la figura 2, se aprecia la estructura.

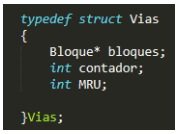

Figura 2: Vías

- **Caché**: Esta estructura es la mas relevante, debido a que contiene a las dos anteriores, y además almacena todo lo que se necesita teóricamente, como también de manera practica. Contiene unas variable de tipo *int* que guarda la cantidad de datos necesarios para poder armar físicamente la memoria caché, además de guardar la cantidad de *Hit* y *Miss* que se obtuvo, y por ultimo, contiene un puntero a la estructura *Vías*, la cual se ocupará de almacenar las vías necesarias para la memoria caché. En la figura 3, se aprecia la estructura.

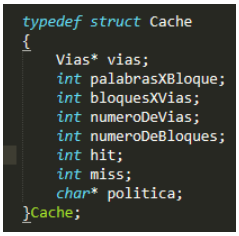

Figura 3: Caché

3.2. Funciones

Las funciones se separan en tres grandes partes, primero entre las que son necesarias para el desarrollo del programa, segundo las que actúan directamente en las distintas políticas de reemplazo y por ultimo las funciones que se encargan de lectura y escritura de los archivos.

3.2.1. Funciones necesarias para el programa

Estas funciones son las que se implementaron antes de abordar el objetivo principal de la solución, que consultar por cada uno de las direcciones de memoria que se encuentran en el archivo leído.

- *void elCache(Cache* cache, char nombre[])*: Esta funcion es la que produce la lógica del programa, ya que se encarga de leer el archivo y a la ves ir aplicando a cada dato leído las técnicas de reemplazos en caso que el caché se encuentre completo, en caso contrario, este es agregado a la memoria caché de forma normal.

- *void colocarPalabra(Cache* cache ,int via,int bloque,int dato)*: Esta función se encarga de colocar el dato correspondiente a la dirección de memoria que se consulto, realizando los cálculos correspondientes para obtener el valor a colocar en la memoria caché.

- *Cache* iniciarCache(char* politica, int vias, int palabras, int bloques)*: Esta función se encarga de inicializar cada una de las variables que contiene la estructura *Cache*, además de dejar seteado cada uno de los datos de la memoria caché en -1, que representa que ese espacio de la cache se encuentra vacío.

- *int validarDatos(int argc, char** argv)*: Esta función se encarga de verificar que los datos entregados al momento de ejecutar sean correctos y cumplan con los requisitos para poder ser tomados como validos.

3.2.2. Funciones para cada política de reemplazo

Las funciones que representan las políticas de reemplazo son las siguientes:

- *void FIFO(Cache* cache, int dato)*: Esta función se encarga de verificar si el dato actual se encuentra en la memoria caché, en ese caso, se produce un aumento a los *Hit*, en caso contrario, se produce un aumenta en los *Miss*, además de realizar un llamado a la función *colocarPalabra* que se encarga de colocar el o los datos correspondientes en la memoria caché.

- *void MRU(Cache* cache, int dato)*: Esta función se encarga de verificar si la memoria caché en primera instancia se encuentra completa, en caso que no lo este, agrega el valor de manera idéntica a la política de reemplazo *FIFO*, en caso que la memoria caché este completa verifica si el datos actual se encuentra en la caché, en ese caso se realiza un aumento a los *Hit*, en caso contrario se realiza un aumento a los *Miss*, además de realizar un llamado a la función *colocarPalabra* que se encarga de colocar el o los datos correspondientes en la memoria caché.

- *void LRU(Cache* cache, int dato)*: Esta función se encarga de verificar si la memoria caché en primera instancia se encuentra completa:

  - Si esta completa: Busca si el valor actual se encuentra en la caché:

    - Si esta en caché: Se realiza un aumento en los *Hit*, además de buscar el índice del bloque actual con la ayuda de la función *indiceBloque*. Finalmente, se realiza el llamado a la función *interacionBloquePolLRU*, que se encarga de aumentar la variable *LRU* de la estructura *Bloque* por cada dato en la cache distinto de -1.
    - Si no esta en caché: Se realiza un aumento en los *Miss*, además de buscar el índice del bloque actual con la ayuda de la función *indiceBloque*, luego se

realiza un llamado a la función *colocarPalabra* que se encarga de colocar el o los datos correspondientes en la memoria caché. Finalmente, se realiza un llamado a la función *interacionBloquePolLRU*, que se encarga de aumentar la variable *LRU* de la estructura *Bloque* por cada dato en la cache distinto de -1.

- No esta completa: Busca si el valor actual se encuentra en la caché:
  - Si esta en caché: Se realiza un aumento en los *Hit*, además de buscar el índice del bloque actual con la ayuda de la función *indiceBloque*. Finalmente se realiza el llamado a la función *interacionBloquePolLRU*, que se encarga de aumentar la variable *LRU* de la estructura *Bloque* por cada dato en la cache distinto de -1.
  - Si no esta en caché: Se realiza el calculo del bloque donde se va a localizar la palabra, además de aumentar los *Miss*, seguido a eso se realiza el llamado a la función *interacionBloquePolLRU*, que se encarga de aumentar la variable *LRU* de la estructura *Bloque* por cada dato en la cache distinto de -1. Finalmente, realiza un aumento en la variable contador que se encuentra en la estructura *Vías*.

3.2.3. Funciones de escritura/lectura de los archivos

Estas funciones están encargadas de leer la información presente en los archivos de entrada, además de escribir la salida del programa.

- *void escribirArchivoCache(Cache* cache, char nombre[])* y *void escribirPorcentajesMissHitt(Cache* cache,char nombre[])*: Estas funciones se encargan de generar los archivos *".out"*, que contienen los datos alojados en la caché posterior a la última consulta realizada a éste en el primer archivo y las estadísticas del caché en el segundo archivo.

4. Experimento

4.1. Resultados obtenidos

El programa consta con una interfaz, la cual el usuario debe ingresar la opción a realizar para poner en curso el programa, además de interactuar insertando los nombre de los archivos que el programa va solicitando para su desarrollo.

En la figura 4, se muestra la salida que entrega el programa al probar el archivo de prueba numero dos:

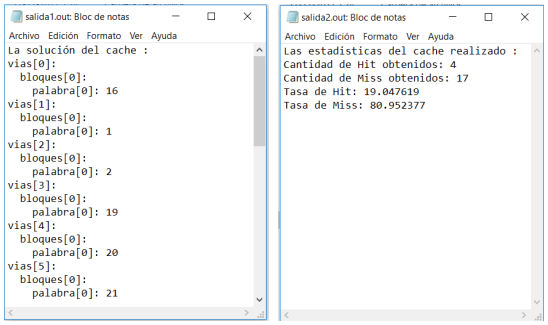

Figura 4: Resultado de ejemplo numero dos.

4.2. Análisis de resultados

A medida que se iba sometiendo a pruebas el programa, el hecho de incrementar la cantidad de palabras por bloques, esto genera un incremente en la tasa *Hit*, así mejorando el rendimiento de la memoria caché, ya que se evita que busque el dato en la memoria principal.

Lo anterior, se aprecia independiente de la política de reemplazo que se utilice, sin embargo, no se omitir que las políticas de reemplazos ayudan a un mejor rendimiento, ya que utilizando la política de reemplazo *MRU* se ha logrado un rendimiento mejor en general, en comparación a las dos políticas de reemplazo restantes, las cuales tiene un rendimiento parecido.

La solución propuesta es escalable, ya que se puede agregar las instrucciones restantes al código fuente. Además, de mencionar que se implemento un algoritmo de

$$O(n) = n^2 \tag{1}$$

5. Conclusiones

En este presente Laboratorio se pudo llevar a practica la teoría que se aprendió en la cátedra, pudiendo tener una visión mas integral del cómo llevar a la práctica cada uno de los elementos vistos y estudiados.

Como se ha mencionado anteriormente, se ha podido lograr la implementación en cada uno de los objetivos tanto general como específicos. Además, destacar que la mayor dificultad dentro del desarrollo del programa fue la organización del código para obtener la solución requerida.

Finalmente, se debe destacar que los resultados del programa fueron exitosos en las pruebas realizadas, además se implementó un algoritmo de

$$O(n) = n^2 \tag{2}$$

sin embargo, se debe dejar abierta la posibilidad de posibles mejoras para dicho algoritmo.

Bibliografía

Guzmán, A., Wladdimiro, D., and Rosas, E. (2017). Organización de computadores jerarquía de memoria. [Online] http://www.udesantiagovirtual.cl/moodle2/pluginfile.php?file=%2F219108%2Fmod_resource%2Fcontent%2F1%2FOC_Cach_.pdf.

LG, M. (04-07-2016). ¿que es el caché? [Online] http://www.lgblog.cl/tecnologia/que-es-el-cache/.