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

Laboratorio Nº 3

Integrantes:	Gabriel Gaete L.
Curso:	Organización de Computadores
Sección	0-L-2
Profesor:	Nestor González & Leonel Medina
Ayudantes:	Ricardo Álvarez & Matías Fuentes

25 de Agosto de 2018

Tabla de contenidos

Índice de figuras

1. Introducción

El presente informe pretende explicar cómo fue solucionado un problema que involucra el concepto de memoria, caché, y políticas de reemplazo, utilizando el lenguaje de programación estructurado C.

En la asignatura de "Organización de computadores", se puede ver que uno de los principales factores que incide en el rendimiento de un programa, es la cantidad de accesos a memoria que debe realizar el procesador para obtener datos en tiempo de ejecución. Con el fin de evitar los accesos a la memoria principal, cuyos tiempos son costosos, se utilizan memorias de tipo caché, las que, en comparación a la memoria principal son de un tamaño menor, sin embargo, dado que estas son ubicadas de una forma cercana al procesador, los tiempos de acceso a esta son considerablemente más rápidos que los accesos a la memoria principal. Sin embargo, dado que la capacidad de estas memorias es reducida, no pueden tener todos los datos de la memoria principal, es que los datos deben ser ingresados por bloques a una dirección determinada de la caché, y en caso de que esa dirección esté ocupada, se debe reemplazar el contenido anterior por el que esté ingresando a la caché, mediante políticas como MRU, LRU, FIFO y LIFO. Conociendo esto, cabe hacerse las siguientes preguntas: ¿qué configuración de caché utilizar? ¿cuál entregará una mayor tasa de éxito?.

1.1. Enunciado del problema

El problema a resolver es el siguiente. Dado un archivo de texto con las direcciones de memoria de los bloques, un tamaño de cache, y una cantidad dada de palabras por bloque, se solicita determinar mediante qué configuración de caché, y qué políticas de reemplazo, se optimiza la cantidad de éxitos o hits en la memoria caché.

1.2. Motivación

La principal motivación para llevar a cabo este laboratorio es comprender la forma en que se organiza una memoria caché, con el fin de optimizar los accesos a memoria.

1.3. Objetivos

1.3.1. Objetivos generales

Desarrollar un programa capaz de determinar la configuración óptima de una memoria caché para un conjunto establecido de direcciones de memoria para los bloques.

1.3.2. Objetivos específicos

Entender la importancia de la organización interna de una memoria caché, junto a las diferentes alternativas existentes para las políticas de reemplazo.

1.4. Herramientas

Las herramientas utilizadas en la implementación de la solución a este laboratorio son las siguientes:

1. Lenguaje de programación C.
2. Uso de estructuras.
3. Uso de memoria dinámica.

1.5. Estructura del informe

El presente escrito detalla a continuación conceptos que fueron la base para llevar a cabo la solución de este problema, tales como *memoria caché*, *políticas de reemplazo*, etc, con el fin de dar a conocer los conceptos investigados, para posteriormente llegar a una descripción de la solución.

2. Marco teórico

2.1. Conocimientos generales previos

A modo de manual, cabe mencionar que para facilitar la compilación del código fuente del programa, se ha facilitado un *makefile*. Para su utilización, se tienen los siguientes comandos:

- *make all* permite la compilación el código fuente.
- *make clean* permite eliminar el archivo ejecutable, luego de la utilización del programa.

Por otra parte, la ejecución del programa se hace de forma manual, de la siguiente forma: ***./main.o -n filename.txt -m cacheBits -p wordsInBlock***

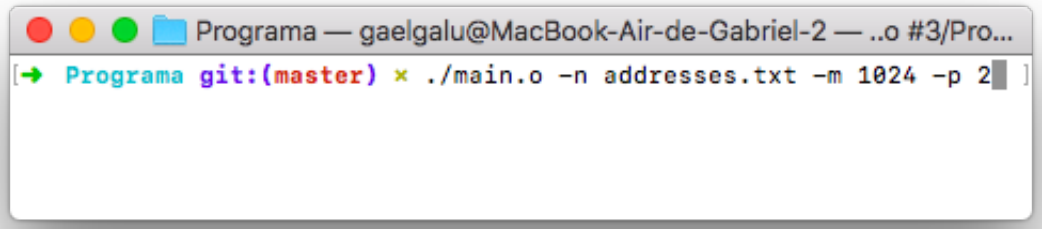

Figura 1: Ejemplo de Ejecución

En la figura 1 se puede apreciar cómo sería una correcta ejecución del programa, donde el archivo con direcciones se llama *addresses.txt*, la caché consta de 1024 bits, mientras que cada bloque se compone de 2 palabras. Cabe considerar, que tanto la memoria de la caché, como la cantidad de palabras por bloque, debe ser un número potencia de dos, de lo contrario, el programa podría no funcionar correctamente.

Por último, cabe considerar que el archivo de entrada DEBE estar en la misma carpeta que el programa ejecutable. Además, se recomienda que este archivo sea de extensión .txt, al igual que el archivo de salida correspondiente a la configuración de la caché. Por otro lado, el archivo con los valores de la memoria debe tener extensión .csv.

2.2. Memoria Caché

La memoria caché es un tipo de memoria volátil, al igual que la memoria RAM, pero muy rápida. Tiene como principal función el almacenar instrucciones y datos a los que el procesador debe acceder continuamente, con la finalidad de que para el procesador, estos datos sean de acceso instantáneo. Su funcionamiento es relativamente simple: cada vez que el procesador quiere acceder a un nuevo dato, éste se almacena en la memoria caché. Entonces, cuando se necesita recurrir nuevamente al mismo dato, el sistema se dirigirá directamente al caché. Este ciclo de almacenamiento y rescate de datos, obliga a la memoria caché a estar en continua renovación, y es aquí donde entran las políticas de reemplazo.

2.3. Políticas de reemplazo

Las políticas de reemplazo son las encargados de seleccionar que bloque de la memoria caché se va a reemplazar cuando ésta está llena y tenemos que asignar un bloque de la memoria principal en la memoria caché. Cada vez que el procesador accede a la memoria caché para buscar un dato y este no se encuentra, se produce un fallo (o *miss*), lo que obliga al procesador a ir a buscar ese dato a la memoria principal, obtener el bloque en el que se encuentra el dato, y llevar ese bloque a la memoria caché. La memoria caché directa (es decir, cuando se tiene un bloque por conjunto), es la única que no tiene políticas de reemplazo, puesto que al haber un solo bloque en cada conjunto, la asignación es directa. Las políticas de reemplazo más conocidas y utilizadas son las siguiente:

- *FIFO (First In First Out)*: El primer bloque en ingresar, es el primer bloque en ser reemplazado.
- *LIFO (Last In First Out)*: El último bloque en ingresar, es el primer bloque en ser reemplazado.
- *LRU (Least Recently Used)*: El bloque menos recientemente usado, es el primer bloque en ser reemplazado.
- *MRU (Most Recently Used)*: El bloque más recientemente usado, es el primer bloque en ser reemplazado.

3. Desarrollo

En primer lugar, el problema presentado en el laboratorio fue dividido en varios sub-problemas, con el fin de ser resuelto por partes. Estos sub-problemas, fueron clasificados de la siguiente forma:

1. Almacenamiento y estructuras para el manejo de datos.
2. Representación de la memoria caché.
3. Implementación de las políticas de reemplazo.
4. Búsqueda del resultado óptimo.

3.1. Almacenamiento y estructuras para el manejo de datos

El primero de los sub-problemas a resolver, se trató de encontrar una forma de almacenar los datos leídos del archivo. Dado que no se conoce la cantidad de números/direcciones que se deben leer en el archivo, se ha implementado una lista enlazada para almacenar estos números.

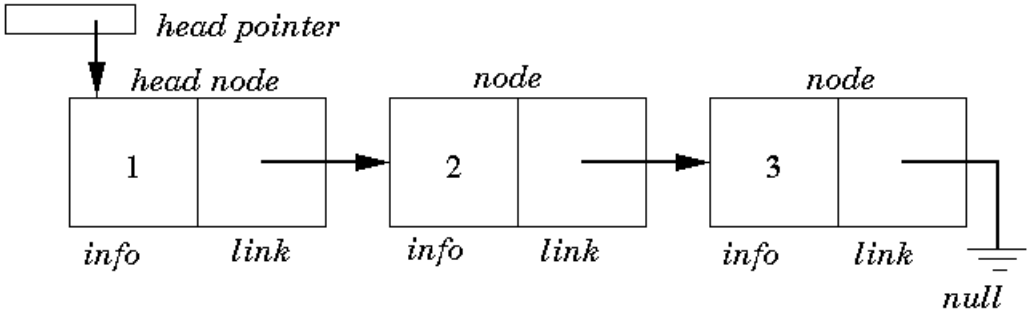

Figura 2: Estructura que almacena las direcciones de los bloques

3.2. Representación de la memoria caché

Para la representación de la memoria caché, se ha implementado una matriz bidimensional de números enteros, en la que cada fila representa un conjunto, mientras que cada columna representa un bloque. Cabe destacar que las palabras no han sido representadas en esta matriz, puesto que se asume la dirección entregada en el archivo como la dirección del bloque, y no de la palabra en si.

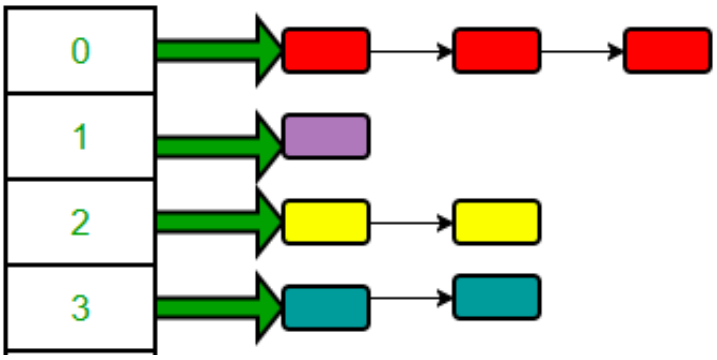

Figura 3: Representación de una memoria caché de 4 conjuntos.

Dado que tanto la cantidad de filas como las de columnas deben ser números potencias de dos, es que se ha optado por representar los espacios vacíos de memoria con valores negativos (-1). Es decir, una caché totalmente vacía, para efectos del diseño de la solución, es una matriz rellena de -1.

3.3. Implementación de políticas de reemplazo

3.3.1. FIFO

Para la implementación de la política de reemplazo FIFO, se ha utilizado una lista enlazada de números. Cada vez que un número es insertado en la memoria caché, se inserta también en esta lista. En caso de que un bloque no pueda ser insertado, dado que ya no queda espacio en su respectivo conjunto de la memoria caché, se itera por sobre la lista enlazada, mientras también se itera por los bloques. El primer bloque que aparezca tanto en la lista enlazada como en el conjunto de la memoria, es el bloque a reemplazar.

3.3.2. LIFO

Para la implementación de la política de reemplazo LIFO, cuando se busca insertar un bloque dentro de un conjunto de la memoria caché y este se encuentra lleno, se reemplaza el último elemento del conjunto, dado que siempre el último elemento de ese conjunto será el último elemento que ha ingresado.

3.3.3. LRU

Para la implementación de la política de reemplazo LRU, se ha implementado una lista enlazada, al igual que en el caso de FIFO. La técnica utilizada es la misma, con la diferencia de que cada vez que se ingresa un número a la lista enlazada, si es que este número ya había sido ingresado antes, se borra su aparición antes de ser ingresado nuevamente. Esto permite que la prioridad de orden de uso vaya quedando al final, es decir, el último de la lista enlazada es el elemento más recientemente utilizado. En caso de necesitar eliminar un bloque de un conjunto, se busca el primer elemento que se encuentre tanto en el conjunto como en la lista enlazada, siendo ese bloque el eliminado.

3.3.4. MRU

Para la implementación de la política de reemplazo MRU, se ha utilizado la misma técnica que para LRU, con la diferencia de que al momento de buscar un elemento dentro de la lista enlazada, se itera desde el último al primero (es decir, se invierte la lista enlazada), con el fin de encontrar el elemento más recientemente usado que esté dentro del conjunto objetivo.

3.4. Búsqueda del resultado óptimo

Para esto, se ha implementado una lista enlazada en la que se almacenan los resultados con sus estadísticas (cantidad de hits, miss, política de reemplazo utilizada, asociatividad y valores en la caché). Cuando se termina de simular una memoria caché, se inserta el resultado en esta lista. Posteriormente, se buscan los resultados con mayor número de hits y se imprime la información y valores en la caché en los respectivos archivos de salida.

4. Experimentos a realizar

Para probar el programa realizado, se tienen los siguientes archivos de prueba:

4.1. Pruebas realizadas

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
1
```

Figura 4: Archivo de entrada 1, contiene las direcciones de bloques.

```
➜ Programa git:(master) × ./main.o -n addresses.txt -m 512 -p 2
[>] Nombre de archivo para la(s) mejor(es) configuracion(es) de cache: config.txt
[>] Nombre de archivo ver el estado final de la cache: cache.csv
Archivo config.txt escrito satisfactoriamente.
Archivo cache.csv escrito satisfactoriamente.
➜ Programa git:(master) ×
```

Figura 5: Ejecución en la terminal.

4.2. Resultados obtenidos

```
0.
Asociatividad de la caché: 4-asociativo
Política de reemplazo: LIFO
Tasa de hits: 0.055556
Tasa de miss: 0.944444

1.
Asociatividad de la caché: 8-asociativo
Política de reemplazo: LIFO
Tasa de hits: 0.055556
Tasa de miss: 0.944444

2.
Asociatividad de la caché: 4-asociativo
Política de reemplazo: MRU
Tasa de hits: 0.055556
Tasa de miss: 0.944444

3.
Asociatividad de la caché: 8-asociativo
Política de reemplazo: MRU
Tasa de hits: 0.055556
Tasa de miss: 0.944444
```

Figura 6: Archivo de salida1, contiene las organizaciones óptimas de la caché.

0.		
Conjunto	Bloque	DirecciV≥n
0	0	0
0	1	1
0	2	2
0	3	16
1	0	4
1	1	5
1	2	6
1	3	15
1.		
Conjunto	Bloque	DirecciV≥n
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	16
2.		
Conjunto	Bloque	DirecciV≥n
0	0	0
0	1	1
0	2	2
0	3	16
1	0	4
1	1	5
1	2	6
1	3	15
3.		
Conjunto	Bloque	DirecciV≥n
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
0	5	5
0	6	6
0	7	16

Figura 7: Archivo de salida2, contiene los valores finales de la caché.

4.3. Análisis de resultados

Se ha logrado que el programa sea capaz de implementar una representación válida para una memoria caché, considerando sus distintos niveles de asociatividad para cada uno de los casos. Además de esto, el programa es capaz de reconocer y filtrar de buena manera los resultados óptimos de caché para las direcciones ingresadas en el archivo. Sin embargo, dado que se ha considerado estas direcciones como una referencia a un bloque, y no una referencia a una palabra, es que las tasas de miss son extremadamente grandes.

5. Conclusiones

Los objetivos indicados al inicio de este informe se han cumplido de manera total, puesto que se ha logrado implementar una estructura de datos que permite representar de manera correcta la organización interna de una memoria caché, permitiendo simular el ingreso de direcciones de memoria referenciando a bloques, y con esto, logrando determinar cuál nivel de organización interna de la memoria caché es óptima dependiendo de diferentes variables, tales como su capacidad, las direcciones de los bloques, etc.

Para futuras experiencias, se recomienda optimizar las tasas de hit, incorporando conceptos como el de localidad espacial (esto puede hacerse referenciando palabras en lugar de bloques). Además se espera mejorar la investigación previa a la implementación de una solución, además de una perfeccionamiento en los algoritmos, los cuales se espera que sean más compactos e intuitivos.

El llevar a cabo este laboratorio ha sido fructífero, sobretodo desde el punto de vista de los conocimientos asimilados. Tener la idea de cómo se optimizan los accesos de memoria a través del uso de una memoria caché, puede ser una herramienta muy eficaz a la hora de comparar diferentes organizaciones y jerarquías de memoria.