Universidad de Buenos Aires

Facultad de Ingeniería

66.20 - Organización de Computadoras

Trabajo Practico 2

Primer cuatrimestre de 2021

Hojman de la Rosa, Joaquin Guido	102264	jhojman@fi.uba.ar
Giampieri Mutti, Leonardo	102358	lgiampieri@fi.uba.ar
Giardina, Fernando	103732	fgiardina@fi.uba.ar

Índice

1. Introducción

El presente informe teórico se realiza teniendo en cuenta el desarrollo práctico correspondiente al Trabajo Práctico N°2 de la materia Organización de Computadoras (66.20).

Reúne algunos ítems principales de la implementación del trabajo, que consiste en desarrollar una simulación del funcionamiento de la memoria caché, en conjunto con la memoria principal, visto en clase, y siendo capaz de proveer ciertas funcionalidades necesarias en la cotidaneidad de esta interconexión, como la lectura y escritura del caché.

1.1. Objetivos

El objetivo del trabajo es simular una memoria cache asociativa por conjuntos, a la cual le podemos pasar por parámetro el número de vías, la capacidad y el tamaño del bloque.

La política de reemplazo que utilizaremos es FIFO y la política de escritura WT/ WA (Write Trought/Non write Allocate).

Se asume que el espacio de direcciones es de 16 bits, y hay entonces una memoria principal a simular con un tamaño de 64KB. Cada bloque de la memoria caché deberá contar con su metadata, incluyendo el bit V, el tag, y un campo que permita implementar la política de FIFO.

Luego de implementado el programa, se ejecutaran una serie de pruebas que mostraran y validaran el correcto funcionamiento del mismo.

2. Detalles de Implementación

2.1. TDA controller: init()

El tipo de dato controller se utiliza para almacenar toda aquella información que es necesaria a la hora de ejecutar los distintos comandos correspondientes a la memoria caché. Entre la información que almacenauarda encontramos:

- u_int ways: almacena la cantidad de vías que va a tener la memoria caché (en bytes).
- u_int cache_size: guarda el tamaño total de la memoria caché (en KB).
- u_int block_size: contiene el tamaño de cada bloque (en bytes).
- char input[MAX_FILE]: almacena el nombre del archivo de donde leer las instrucciones.
- char output[MAX_FILE]: guarda el nombre del archivo donde escribir los resultados.
- bool print_version: guarda si debe imprimir la versión o no.
- bool print_help: guarda si debe imprimir la ayuda o no.

En el init se inicializan todos estos campos con los valores adecuados según haya indicado el usuario. Primero analiza si se encuentra en el caso en donde solo debe imprimir la ayuda o la version, es decir, cuando se da:

```
tp2 -h
tp2 -V
```

Si se introduce un argumento que no es ni -h o -V devuelve el codigo de error apropiado. Si no se encuentra frente a ninguno de estos casos, se encuentra frente a un intento de ejecutar distintas instrucciones para escribir/leer de la memoria.

A continuación itera los argumentos interpretandolos. Para eso se diseño una función interpret_argument(controller_t *controller, char const *arg, bool *next_arg) para comparar los argumentos con los que el programa conoce e inicializar los campos del controller_t adecuadamente. bool *next_arg es un vector de booleanos, el cual tiene todos sus elementos en false inicialmente, y, en caso de encontrar alguna instrucción conocida, como por ejemplo w"que representa la cantidad de vías, pone un campo en true, para que la siguiente iteración tenga el significado de cargar efectivamente la cantidad de vías a leer. Los campos de este vector de booleanos son: output –¿0, ways–¿1, cachesize–¿2, blocksize–¿3, input–¿4. Otra ejemplo de implementación es la del -o. Cuando encuentra dicho comando, se guarda en la posición 0 del vector next_arg el valor "true". De esta forma en la proxima iteración se interpreta el argumento como el nombre del archivo. El archivo es por default stdout.

Si en la lectura de argumentos alguno es erroneo se devuelve un error especifico. Una vez terminado el init ya se tiene un controller que sabe todo lo necesario para ejecutar el algoritmo.

2.2. TDA controller: controller_execute()

El objetivo de esta función es llamar al TDA encargado de ejecutar la lectura del archivo con los argumentos pertinentes y escribir su resultado en donde el usuario indicó.

Primero se revisa si se encuentra en el caso en el que solo debe imprimir la ayuda o la versión. Si se encuentra en ese caso, llama una función que imprime lo que debe y retorna de la función.

Una vez hecha esa revisión, sabe que se encuentra en el caso en el que debe llamar a las funciones de lectura de archivo que permitan interpretar lo que se quiere escribir/leer en memoria. Para ello inicializa un file, que luego se explicará como funciona más detalladamente, pero basicamente, lee una instrucción, la interpreta, accede al TDA caché y luego imprime por pantalla el resultado. A continuación se coloca el código del controller_execute(), en el que además, se pueden observar, que se reciben los resultados de los llamados a las funciones, para poder luego manejar los errores, como se explicará en el siguiente apartado:

```
        if(controller->print_help){
                print_help();
                return SUCCESS;
        } else if(controller->print_version) {
                print_version();
                return SUCCESS;
        }
        int file_result;
        file_t file;
        if(strncmp("-", controller->output, MAX_FILE)==0)
                file_result = file_init(&file,
                controller->input, NULL);
        else
                file_result = file_init(&file,
                controller->input, controller->output);
        if (file_result != SUCCESS) return file_result;

        file_result = file_iterate(&file);

        if (file_result != SUCCESS) return file_result;

        file_result = file_uninit(&file);

        if (file_result != SUCCESS) return file_result;

        return SUCCESS;
```

2.3. TDA controller: error_handler()

La función se encarga de recibir un número que representa un código de error e imprime un mensaje por stderr relacionado con lo que falló, para de esa forma informar al usuario. Entre los errores reconocidos:

- MISSING_ARGUMENT: Faltan uno o más argumentos obligatorios (-w, -cs, -bs).
- NOT_HELP_NOR_VERSION: Se envia un solo arguemento que no es ni -h ni -V.
- UNKNOWN_COMMAND: Comando enviado no reconocido.
- FILE_ERROR: Se produjo un error al intentar interpretar alguna instrucción del input file.
- ARGV_QUANTITY: No se ingresó una cantidad correspondientes de argumentos (el mínimo es 2 y el máximo es 10), contando como un argumento el "./tp2".
- SHOULDNT_REACH_HERE: Se llega a una situacion a la que no se tendria (es un error de diseño de codigo).

2.4. TDA File: file_init() y file_uninit()

La función file_init() recibe dos strings, en primer lugar, el correspondiente al nombre del archivo a leer, en segundo lugar, el nombre del archivo a escribir. Éstos se utilizan para poder abrir el archivo correspondiente ya sea para lectura/escritura. En el caso de que alguno de los dos strings recibidos tenga el valor NULL, se utilizará como archivo de lectura la entrada o salida estándar según corresponda. A su vez, el método file_uninit(), se encarga de cerrar los archivos abiertos en caso de ser necesario (si no corresponden a stdin/stdout).

2.5. TDA File: parse_int()

Esta función se utiliza para interpretar, dentro de un buffer, valores numéricos. Recibe dos caracteres, que representan los valores iniciales y finales, entre los cuales estará comprendido el valor numérico. Posteriormente, mediante técnicas de aritméticas de punteros, guarda en un buffer temporal el valor deseado. Este valor, al estar representado en forma de vector de chars, al ser devuelto de la función parse_int, se le aplica la función atoi(), para que sea efectivamente sea un entero lo que se devuelve, y pueda sea interpretado como tal, posteriormente.

2.6. TDA File: file_iterate()

La función file_iterate() se encarga principalmente de recorrer el archivo de lectura, y, cada vez que se lee una línea (mediante el comando getline), ésta es parseada de tal forma que, según el tipo de instrucción que se desee ejecutar, se invoquen a los métodos correspondientes de la memoria caché. Para el correcto funcionamiento del programa, se invoca siempre, en primer lugar, a la inicialización del TDA caché (init()), esto sucede independientemente de que sea indicado explícitamente en el archivo o no. Además, en los casos de lectura o escritura, para saber si se ha producido un hit o miss, se calcula inicialmente la cantidad de HIT ocurridos previo a la ejecución de la instrucción, y posteriormente, en caso de resultar la cantidad idéntica a la disponible luego de la ejecución de la función, se asume que lo que realmente hubo fue un MISS, caso contrario, se produjo un HIT. Esto es informado al usuario.

Resumiendo, las instrucciones válidas que puede contener el archivo de lectura son las siguientes:

- Inicialización (init): Se invoca la inicialización del TDA caché.

- Escritura (W): Se escribe el byte deseado en la posición correspondiente mediante el llamado a la función write_byte(), y se muestra si fue un HIT o MISS.
- Lectura (R): Se presenta el valor obtenido de la dirección deseada, que es el valor que devuelve la función read_byte(). Además, se muestra si hubo HISS O MISS.
- Miss Rate (MR): Se muestra el resultado de la invocación a la función get_miss_rate().

2.7. TDA Caché: init()

Para comenzar con los detalles de implementación del TDA principal iniciaremos con la explicación de lo que hace su init(). Su implementación es bastante simple y lo que se hace es calcular la cantidad de sets de la memoria a partir de la división de la cantidad de bytes totales y las vías y además se encarga de reservar la memoria necesaria para todos los bloques. Luego llena a la memoria principal con ceros e inicializa algunos campos de todos los bloques como por ejemplo pone todos los bloques en inválidos. Análogamente, se tiene un destroy() que libera la memoria reservada por lo que es importante llamarlo antes de terminar la ejecución del programa.

2.8. TDA Cache: read_block()

La siguiente funcion es una de las más importantes del trabajo ya que su función es fundamental: se encarga de leer un bloque de memoria. Si ya se encuentra en la memoria cache entonces no se hace nada, pero si no se encuentra se lo carga en la posición correspondiente. A continuación mostraremos su implementación:

```
void read_block(int blocknum) {
    int address = blocknum << (int) log2(block_size);
    int hit_position = hit_blocknum(address);

    if (hit_position == -1) { // NO HIT
        move_block_to_cache(blocknum);
        cache_memory.miss_amount++;
    } else { // HIT
        cache_memory.hit_amount++;
    }
}
```

Primero obtiene la dirección de memoria a la que comienza el bloque y se fija a través de la función hit() si ya se encuentra en la memoria cache. Si no es así, mueve el bloque de la memoria principal a cache. En caso contrario, de hit, se actualizan los datos asociados a FIFO. Como si se llaman funciones cuya implementación es importante, entonces las mostraremos. Comenzaremos con hit_blocknum():

```
int hit_blocknum(int address) {
    unsigned int set = find_set(address);
    unsigned int tag = find_tag(address);

    unsigned int blocknum;
```

```
    for (int i = 0; i < ways; i++) {
        blocknum = cache_block_number(i, (int) set);
        if (cache_memory.memory[blocknum].V == 1 &&
            cache_memory.memory[blocknum].tag == tag) {
            return (int) (blocknum);
        }
    }

    return -1;
}
```

La función se encarga de devolver el bloque de la memoria cache en el que se encuentra la dirección de memoria si así lo hace, si no es así, devuelve -1 para indicar que el bloque no está cargado en memoria cache. Lo más importante de su implementación es el loop. Lo que hace es recorrer todo el bloque correspondiente a la dirección buscada y si el tag coincide significa que nos encontramos frente al bloque buscado por lo que se devuelve el número del bloque en memoria cache. Si se sale del loop significa que no se encontró la dirección por lo que se devuelve -1.

Luego continuaremos con la explicación de la implementación de la función move_block_to_cache() y comenzaremos mostrando su implementación:

```
void move_block_to_cache(int blocknum) {
    int address = blocknum << (int) log2(block_size);
    int spot = find_spot(address);

    memcpy(cache_memory.memory[spot].data,
           &principal_memory[address],
           block_size);

    cache_memory.memory[spot].tag = find_tag(address)& 0xFFF;

    last_in_update(spot);
}
```

Como su nombre indica, la función se encarga de mover de la memoria principal a cache un bloque. Para ello obtiene el número de bloque de cache con find_spot(). No explicaremos su implementación muy a fondo , pero basicamente hace lo siguiente: el memcpy copia el nuevo bloque a cache y por último se actualiza el FIFO al igual que el caso donde había hit.

La función read_byte() se encarga de leer el byte pedido. Su implementación es sencilla. Llama a read block para copiar, si no está, el bloque que contiene la dirección requerida en memoria cache. Luego simplemente busca el byte a partir de su offset en el bloque (que se obtiene con la dirección) y se devuelve el byte. En código se ve de la siguiente forma:

```
unsigned char read_byte(int address, char * hit) {
    size_t temp_hit_amount = cache_memory.hit_amount;
    read_block(address >> (int) log2(block_size));
    if (cache_memory.hit_amount == temp_hit_amount+1){
        *hit = '1';
```

```
    }else{
        *hit = '0';
    }

    // Hay hit por el read_block anterior
    int slot = hit_blocknum(address);
    return cache_memory.memory[slot].data[get_byte_offset((u_int)

    address)];
}
```

2.9. TDA Cache: write_byte()

Esta función se encarga de escribir un byte en la dirección pedida, valiendose de la politica de escritura pedida: Write Trought y Non Write Alocate. Su implementación es sencilla y luce de la siguiente manera:

```
char write_byte(int address, unsigned char value) {
    size_t temp_hit_amount = cache_memory.hit_amount;
    read_block(address >> (int) log2(block_size));

    // Hay hit por el read_block anterior
    int slot = hit_blocknum(address);
    cache_memory.memory[slot].data[get_byte_offset
    ((u_int) address)]= value;

    write_byte_tomem(address, value);

    if (cache_memory.hit_amount == temp_hit_amount+1){
        return '1';
    }else{
        return '0';
    }
}
```

2.10. Tests

En esta sección se mostrarán las pruebas realizadas sobre las funciones implementadas

./tp2 -w 4 -cs 8 -bs 16 prueba1.mem
Inicialización
Escritura del valor 255 en la posición 0: MISS
Lectura del valor 255 en la posicion 0: HIT
Escritura del valor 254 en la posición 16384: MISS
Lectura del valor 254 en la posicion 16384: HIT
Escritura del valor 248 en la posición 32768: MISS
Lectura del valor 248 en la posicion 32768: HIT

Escritura del valor 96 en la posición 49152: MISS
Lectura del valor 96 en la posicion 49152: HIT
Lectura del valor 255 en la posicion 0: HIT
Escritura del valor 1 en la posición 0: HIT
Lectura del valor 254 en la posicion 16384: HIT
Escritura del valor 163 en la posición 16384: HIT
Lectura del valor 248 en la posicion 32768: HIT
Escritura del valor 32 en la posición 32768: HIT
Lectura del valor 96 en la posicion 49152: HIT
Escritura del valor 49 en la posición 49152: HIT
Miss Rate: 25

./tp2 -w 4 -cs 8 -bs 16 prueba2.mem
Inicialización
Escritura del valor 123 en la posición 0: MISS
Escritura del valor 233 en la posición 1: HIT
Escritura del valor 34 en la posición 2: HIT
Lectura del valor 123 en la posicion 0: HIT
Lectura del valor 233 en la posicion 1: HIT
Lectura del valor 34 en la posicion 2: HIT
Escritura del valor 1 en la posición 0: HIT
Escritura del valor 2 en la posición 1: HIT
Escritura del valor 3 en la posición 2: HIT
Lectura del valor 1 en la posicion 0: HIT
Lectura del valor 2 en la posicion 1: HIT
Lectura del valor 3 en la posicion 2: HIT
Miss Rate: 8

./tp2 -w 4 -cs 8 -bs 16 prueba3.mem
Inicialización
Escritura del valor 1 en la posición 128: MISS
Escritura del valor 2 en la posición 129: HIT
Escritura del valor 3 en la posición 130: HIT
Escritura del valor 4 en la posición 131: HIT
Lectura del valor 1 en la posicion 128: HIT
Lectura del valor 2 en la posicion 129: HIT
Lectura del valor 3 en la posicion 130: HIT
Lectura del valor 4 en la posicion 131: HIT
Lectura del valor 0 en la posicion 1152: MISS
Lectura del valor 0 en la posicion 2176: MISS
Lectura del valor 0 en la posicion 3200: MISS
Lectura del valor 0 en la posicion 4224: MISS
Lectura del valor 1 en la posicion 128: HIT
Lectura del valor 2 en la posicion 129: HIT
Lectura del valor 3 en la posicion 130: HIT
Lectura del valor 4 en la posicion 131: HIT
Miss Rate: 31

```
./tp2 -w 4 -cs 8 -bs 16 prueba4.mem
Inicialización
Escritura del valor 256 en la posición 0: MISS
Escritura del valor 2 en la posición 1: HIT
Escritura del valor 3 en la posición 2: HIT
Escritura del valor 4 en la posición 3: HIT
Escritura del valor 5 en la posición 4: HIT
Lectura del valor 0 en la posicion 0: HIT
Lectura del valor 2 en la posicion 1: HIT
Lectura del valor 3 en la posicion 2: HIT
Lectura del valor 4 en la posicion 3: HIT
Lectura del valor 5 en la posicion 4: HIT
Lectura del valor 0 en la posicion 4096: MISS
Lectura del valor 0 en la posicion 8192: MISS
Lectura del valor 0 en la posicion 0: HIT
Lectura del valor 2 en la posicion 1: HIT
Lectura del valor 3 en la posicion 2: HIT
Lectura del valor 4 en la posicion 3: HIT
Lectura del valor 5 en la posicion 4: HIT
Miss Rate: 17
```

```
./tp2 -w 4 -cs 8 -bs 16 prueba5.mem
Inicialización
Violación de segmento (‘core’ generado)
```

¿Qué ocurrió? ¿Por qué el programa nos devuelve un Segmentation Fault?

Si observamos la prueba, vemos se trata de hacer una lectura de la posición 128 KB, mientras que la memoria cache tiene solamente 64 KB de capacidad, y la memoria principal tambien ocupa 64 KB. Entonces efectivamente es correcto que el programa tire el error de violacion de segmento, ya que se quiere acceder a una posición inexistente.

3. Conclusiones

El trabajo practico 2 consistio en implementar una simulación del funcionamiento de una memoria cache, y podemos afirmar que se logró satisfactoriamente este objetivo.

Consideramos fuertemente que haber realizado el trabajo nos aportó un mayor entendimiento en la lógica de funcionamiento una memoria cache y nos sirvió para constatar que situaciones que sabemos resolver en papel también se pudieron resolver mediante el programa creado, con resultados identicos o muy similares.

Lo mas complejo del trabajo practico fueron las operaciones lógico matemáticas que hubo que realizar, como por ejemplo obtener el set al que pertenece una dirección u obtener la dirección a partir de un bloque de memoria cache

Destacamos la calidad del set de pruebas, que nos ayudó a encontrar el camino correcto mostrándonos la importancia de testear el código correctamente para validar que el funcionamiento del mismo fuera el esperado.

Concluimos entonces que el trabajo se llevo a cabo exitosamente y cumplio su rol educativo para con esta materia.

4. Apendice

4.1. main.c

```
#include <stdio.h>
#include "controller.h"

#define MIN_ARGS 2
#define MAX_ARGS 10

int main(int argc, char const *argv[]){
        if (argc < MIN_ARGS || argc > MAX_ARGS) return
        (int)error_handler(ARGV_QUANTITY);

        controller_t controller;

        size_t controller_ret, execute_ret;
        controller_ret = controller_init(&controller, argc, argv);
        if(controller_ret != 0){
                error_handler(controller_ret);
                return -1;
        }

        execute_ret = controller_execute(&controller);
        if(execute_ret != 0){
                error_handler(execute_ret);
                return -1;
        }

        controller_destroy(&controller);

        return 0;
}
```

4.2. file.h

```
#ifndef FILE_H
#define FILE_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "cache.h"

typedef struct file {
    FILE *file_reader;
```

```
    FILE *file_writer;
} file_t;

size_t file_init(file_t *self,
                 const char *file_name_reader,
                 const char *file_name_writer);

size_t file_uninit(file_t *self);

//Realiza las iteraciones en la lectura del archivo e
//imprime en el archivo correspondiente el resultado
size_t file_iterate(file_t *self);

#endif
```

4.3. file.c

```
#include "file.h"

#define FILE_ERROR 4
#define SUCCESS 0
#define BUF_SIZE 60
#define OPEN_MODE_READ "rb"
#define OPEN_MODE_WRITE "a"

size_t file_init(file_t* self,
        const char* file_name_reader,
        const char* file_name_writer){
        self->file_reader = file_name_reader != NULL ?
                fopen(file_name_reader,OPEN_MODE_READ) : stdin;

        self->file_writer = file_name_writer != NULL ?
                fopen(file_name_writer,OPEN_MODE_WRITE) : stdout;

        if ((self->file_reader==NULL) ||
        (self->file_writer==NULL)){
                return FILE_ERROR;
        }else{
                return SUCCESS;
        }
}

size_t file_uninit(file_t* self){
        size_t result_reader = 0;
        size_t result_writer = 0;
        if(self->file_reader != stdin)
```

```
                result_reader = (size_t)fclose
                (self->file_reader);
        if(self->file_writer != stdout)
                result_writer = (size_t)fclose
                (self->file_writer);
        if ((result_reader == FILE_ERROR) ||
        (result_writer == FILE_ERROR)){
                return FILE_ERROR;
        }else{
                return SUCCESS;
        }
}

int parse_int(char ** buf, const char first_char,
const char second_char){
        int i=0;
        char temp[20];
        char * pointer=*buf+1;
        while(*pointer != first_char)
                pointer=*buf+i++;
        i=0;
        while(*pointer != second_char)
                temp[i++]=*(++pointer);
        temp[i]='\0';
        return atoi(temp);

}

size_t file_iterate(file_t* self){
        char * buf[BUF_SIZE];
        for (int i = 0; i < BUF_SIZE; i++) buf[i]=NULL;
        size_t status=0, read=0;
    init(); //Necesitamos inicializar el TDA siempre.
        fprintf(self->file_writer, "Inicializaci n\n");
        while ((read = (size_t)getline
        (buf, &read, self->file_reader)) != -1) {
        if (strncmp(*buf, "init\n", BUF_SIZE)==0){
                continue;
        }else if(**buf=='W'){
                int dir = parse_int(buf, ' ', ',');
                int value = parse_int(buf, ',', '\n');
                size_t temp = cache_memory.hit_amount;
                write_byte(dir, (unsigned char) value);
                (temp == cache_memory.hit_amount)?
                        strcpy(*buf, "MISS\0"):
                        strcpy(*buf, "HIT\0");
```

```
                        fprintf(self->file_writer,
                         "Escritura del valor %i en la
                         posici n %i: %s\n", value, dir, *buf);
        }else if(**buf=='R'){
                int dir = parse_int(buf, ' ', '\n');
                char hit;
                unsigned char result = read_byte(dir, &hit);
                (hit=='0')?
                        strcpy(*buf, "MISS\0"):strcpy(*buf,
                        "HIT\0");
                        fprintf(self->file_writer,
                                "Lectura del valor %i en la
                                posicion %i: %s\n", result, dir,
                                *buf);
        }else if (strncmp(*buf, "MR\n", BUF_SIZE)==0){
                        fprintf(self->file_writer, "Miss Rate:
                        %i\n", get_miss_rate());
        }else{
                destroy();
                return FILE_ERROR;
        }
    }
    free(*buf);
    destroy();
    return status;
}
```

4.4. controller.h

```
#ifndef CONTROLLER_H
#define CONTROLLER_H

#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define MAX_FILE 50


#define SUCCESS 0
#define MISSING_ARGUMENT 1
#define NOT_HELP_NOR_VERSION 2
#define UNKNOWN_COMMAND 3
#define FILE_ERROR 4
#define ARGV_QUANTITY 5
```

```
#define SHOULDNT_REACH_HERE 6

#define WAYS 0
#define CACHE_SIZE 1
#define BLOCK_SIZE 2
#define OUTPUT 3
#define PRINT_HELP_OR_VERSION 4
#define DONT_KNOW 5



typedef struct controller{
        u_int ways;
        u_int cache_size;
        u_int block_size;
        char input[MAX_FILE];
        char output[MAX_FILE];
        bool print_version;
        bool print_help;
}controller_t;


//Inicializo el TDA controller.
size_t controller_init(controller_t * controller, int argc,
char const *argv[]);

//Destruye el TDA liberando la memoria correspondiente.
void controller_destroy(controller_t * controller);

//Informa que se han ingresado argumentos inv lidos.
size_t error_handler(size_t what);

//Ejecuta el programa con los parametros inicilizados.
size_t controller_execute(controller_t * controller);

#endif
```

4.5. controller.c

```
#include "controller.h"
#include "file.h"
#include "cache.h"
#include <string.h>
#include <limits.h>
#include <stdlib.h>
```

```
#define MIN_ALLOWED 2
#define MAX_OPTION 50

size_t interpret_argument(controller_t *controller,
char const *arg,
        bool * next_arg){
        if((strncmp("-V", arg, MAX_OPTION)==0) ||
           (strncmp("--version", arg, MAX_OPTION)==0)){
                controller->print_version = true;
                return SUCCESS;
        } else if (((strncmp("-h", arg, MAX_OPTION)==0) ||
                (strncmp("--help", arg,MAX_OPTION)==0))){
                controller->print_help = true;
                return SUCCESS;
        } else if (((strncmp("-o", arg, MAX_OPTION)==0) ||
            (strncmp("--output", arg, MAX_OPTION)==0))){
                next_arg[0] = true;
                return SUCCESS;
        } else if (((strncmp("-w", arg, MAX_OPTION)==0) ||
                (strncmp("--ways", arg, MAX_OPTION)==0))){
                next_arg[1] = true;
                return SUCCESS;
        } else if (((strncmp("-cs", arg, MAX_OPTION)==0) ||
                (strncmp("--cachesize", arg, MAX_OPTION)==0))){
                next_arg[2] = true;
                return SUCCESS;
        } else if (((strncmp("-bs", arg, MAX_OPTION)==0) ||
                (strncmp("--blocksize", arg, MAX_OPTION)==0))){
                next_arg[3] = true;
                return SUCCESS;
        } else {
                if(next_arg[0]){
                        strcpy(controller->output, arg);
                        next_arg[0] = false;
                        return SUCCESS;
                } else if(next_arg[1]){
                        controller->ways = (u_int)
                        strtoul(arg, NULL, 10);
                        next_arg[1] = false;
                        return SUCCESS;
                } else if(next_arg[2]){
                        controller->cache_size = (u_int)
                        strtoul(arg, NULL, 10);
                        next_arg[2] = false;
                        return SUCCESS;
                } else if(next_arg[3]){
```

```
                        controller->block_size = (u_int)
                        strtoul(arg, NULL, 10);
                        next_arg[3] = false;
                        return SUCCESS;
                } else if(next_arg[4]){
                        strcpy(controller->input, arg);
                        next_arg[4] = false;
                        return SUCCESS;
                } else {
                        return UNKNOWN_COMMAND;
                }
        }
}

size_t controller_init(controller_t * controller, int argc,
char const *argv[]){
        controller->print_version = false;
        controller->print_help = false;
        controller->ways = 0;
        controller->cache_size = 0;
        controller->block_size = 0;
        strcpy(controller->output,"-");
        //POR DEFAULT SE IMPRIME EN STDOUT

        if(argc == 2){
                controller->print_version = ((strncmp("-V",
                argv[1], MAX_OPTION)==0) ||

                controller->print_help = ((strncmp("-h",
                argv[1], MAX_OPTION)==0) ||
                (strncmp("--help", argv[1], MAX_OPTION)==0));
                if(!controller->print_version &&
                !controller->print_help){
                        return NOT_HELP_NOR_VERSION;
                } else {
                        return SUCCESS;
                }
        } else {
                bool next_arg[4];
                //output --> 0, ways-->1, cachesize-->2,
                //blocksize-->3, input-->4
                for (int i = 0; i < 4; i++)
                        next_arg[i]=false;
                for (int i = 1; i < argc; i++) {
                        if (i==argc-1) next_arg[4] = true;
                        size_t possible_error =
```

```
                        interpret_argument(controller,
                        argv[i],next_arg);
                        if(possible_error != SUCCESS)
                                return possible_error;
                }
                if ((controller->ways==0) ||
                        (controller->cache_size==0) ||
                        (controller->block_size==0))
                        return MISSING_ARGUMENT;
        }
        return SUCCESS;
}

void print_version(){
        fprintf(stdout, "Version 1.0\nOrganizaci n de
        Computadoras - FIUBA\n");
}

void print_help(){
        fprintf(stdout, "Usage:\n\ttp2 -h\n\ttp2 -V\n\ttp2
        [options] M N\n");
        fprintf(stdout, "Options:\n\t-h, --help     Imprime
        ayuda.\n");
        fprintf(stdout, "\t-V, --version  Versi n del programa.
        \n");
        fprintf(stdout, "\t-o, --output   Archivo de salida.\n");
        fprintf(stdout, "\t-w, --ways  Cantidad de v as.\n");
        fprintf(stdout, "\t-cs, --cachesize   Tama o del cach
        en kilobytes.\n");
        fprintf(stdout, "\t-bs, --blocksize   Tama o del bloque
        en bytes.\n");
        fprintf(stdout, "Examples:
        \n\ttp2 -w 4 -cs 8 -bs 16 prueba1.mem\n");
}

size_t error_handler(size_t what){
        switch(what){
                case SUCCESS:
                        break;
                case MISSING_ARGUMENT:
                        fprintf(stderr, "There are missing
                        arguments. Check carefully needed
                        arguments.\n");
                        print_help();
                        break;
                case NOT_HELP_NOR_VERSION:
```

```
                        fprintf(stderr, "Not asked for help nor
                        version and only sent one argument.\n");
                        print_help();
                        break;
                case UNKNOWN_COMMAND:
                        fprintf(stderr, "A command sent is
                        unknown.\n");
                        print_help();
                        break;
                case FILE_ERROR:
                        fprintf(stderr, "
                        ----------------------------\n");
                        fprintf(stderr, "The file couldn't be
                        read properly. Check it and retry.\n");
                        print_help();
                        break;
                case ARGV_QUANTITY:
                        fprintf(stderr, "You've entered too many
                        arguments.\n");
                        print_help();
                        break;
                case SHOULDNT_REACH_HERE:
                        fprintf(stderr, "Programmer's fault.\n");
                        print_help();
                        break;
                default:
                        fprintf(stderr, "There was an unknown
                        error\n");
                        print_help();
        }

        return SUCCESS;
}

size_t controller_execute(controller_t * controller){
        if(controller->print_help){
                print_help();
                return SUCCESS;
        } else if(controller->print_version) {
                print_version();
                return SUCCESS;
        }

        size_t file_result;
        file_t file;
```

```
        cache_memory_size = controller->cache_size;
        block_size = controller->block_size;
        ways = controller->ways;


        if(strncmp("-", controller->output, MAX_FILE)==0)
                file_result = file_init(&file, controller->input,
                NULL);
        else
                file_result = file_init(&file, controller->input,
                controller->output);

        if (file_result != SUCCESS) return file_result;

        file_result = file_iterate(&file);

        if (file_result != SUCCESS) return file_result;

        file_result = file_uninit(&file);

        if (file_result != SUCCESS) return file_result;

        return SUCCESS;
}

void controller_destroy(controller_t * controller){

}
```

4.6. cache.h

```
#ifndef TP2_OC_CACHE_H
#define TP2_OC_CACHE_H

#include <stdlib.h>

#define PRINCIPAL_MEMORY_SIZE 65536

typedef struct block {
    unsigned char *data;
    int fifo_value;
    unsigned int tag: 12;
    unsigned int V: 1;
} Block;

typedef struct cache {
```

```
    Block *memory;
    size_t block_amount;
    size_t set_amount;
    size_t hit_amount;
    size_t miss_amount;
} Cache;

Cache cache_memory;
unsigned char principal_memory[PRINCIPAL_MEMORY_SIZE];
unsigned int cache_memory_size;
unsigned int block_size;
unsigned int ways;

void init();

unsigned int find_set(int address);

unsigned int find_earliest(int setnum);

void read_block(int blocknum);

unsigned char read_byte(int address, char * hit);

void write_byte_tomem(int address, unsigned char value);

char write_byte(int address, unsigned char value);

int get_miss_rate();

void destroy();

#endif //TP2_OC_CACHE_H
```

4.7. cache.c

```
#include "cache.h"
#include <math.h>
#include <stdio.h>
#include <string.h>

void set_all_principal_memory_zero() {
    for (int i = 0; i < PRINCIPAL_MEMORY_SIZE; i++)
        principal_memory[i] = 0;
}

void initialize_blocks() {
```

```
    for (int i = 0; i < cache_memory.block_amount; i++) {
        cache_memory.memory[i].data = malloc(block_size);
        if (cache_memory.memory[i].data == NULL)
            return;
        cache_memory.memory[i].V = 0;
        cache_memory.memory[i].fifo_value = 0;
    }
}

void init() {
    cache_memory.block_amount = (u_int)(
            cache_memory_size * pow(2, 10) / block_size);
    cache_memory.set_amount = cache_memory.block_amount / ways;
    cache_memory.memory = malloc(
    cache_memory.block_amount * sizeof(Block));

    if (cache_memory.memory == NULL)
        return;

    set_all_principal_memory_zero();
    initialize_blocks();
}

unsigned int find_set(int address) {
    // Asumo que los bytes por bloques son multiplo de 2
    unsigned int blocknum = (u_int) address >>
    (int) log2(block_size);
    return (u_int)(blocknum % cache_memory.set_amount);
}

unsigned int cache_block_number(int way, int setnum) {
    return (u_int) setnum * ways + (u_int) way;
}

void last_in_update(int new_blocknum) {
    int prev_fifo_value = cache_memory.memory[new_blocknum].
    fifo_value; // 0 if new
    int blocknum;
    for (int i = 0; i < ways; i++) {
        blocknum = (int) cache_block_number(i,
                                            (new_blocknum -
                                             new_blocknum
                                             % (int) ways) /
                                            (int) ways);
        if (cache_memory.memory[blocknum].V == 1) {
            if (blocknum == new_blocknum) {
```

```
                cache_memory.memory[blocknum].fifo_value = 1;
            } else {
                if (cache_memory.memory[blocknum].fifo_value
                < prev_fifo_value ||
                    prev_fifo_value == 0)
                    cache_memory.memory[blocknum].fifo_value++;
            }
        }
    }
}

unsigned int find_earliest(int setnum){
    for (int i = 0; i < ways; i++) {
        if (cache_memory.memory[cache_block_number(i, setnum)].
        fifo_value ==
            ways)
            return cache_block_number(i, setnum);
    }
    return 0; // Should not happen
}

unsigned int find_tag(int address) {
    return (u_int)(address >> (int) log2(block_size))
            >> (int) log2((double) cache_memory.set_amount);
}

// returns -1 if is not hit, otherwise returns blocknum
int hit_blocknum(int address) {
    unsigned int set = find_set(address);
    unsigned int tag = find_tag(address);

    unsigned int blocknum;
    for (int i = 0; i < ways; i++) {
        blocknum = cache_block_number(i, (int) set);
        if (cache_memory.memory[blocknum].V == 1 &&
            cache_memory.memory[blocknum].tag == tag) {
            return (int) (blocknum);
        }
    }

    return -1;
}

int find_spot(int address) {
    unsigned int set = find_set(address);
```

```
    for (int i = 0; i < ways; i++) {
        unsigned int blocknum = cache_block_number(i, (int) set);
        if (cache_memory.memory[blocknum].V == 0) {
            cache_memory.memory[blocknum].V = 1;
            return (int) (blocknum);
        }
    }

    return (int) find_earliest((int) set);
}

void move_block_to_cache(int blocknum) {
    int address = blocknum << (int) log2(block_size);
    int spot = find_spot(address);
    memcpy(cache_memory.memory[spot].data,
           &principal_memory[address],
           block_size);

    cache_memory.memory[spot].tag = find_tag(address) & 0xFFF;

    last_in_update(spot);
}

void read_block(int blocknum) {
    int address = blocknum << (int) log2(block_size);
    int hit_position = hit_blocknum(address);

    if (hit_position == -1) { // NO HIT
        move_block_to_cache(blocknum);
        cache_memory.miss_amount++;
    } else { // HIT
        cache_memory.hit_amount++;
    }
}

int get_address(int way, int setnum) {
    int blocknum = (int) cache_block_number(way, setnum);
    int address = ((cache_memory.memory[blocknum].tag
            << (u_int) log2((double) cache_memory.set_amount))
            + setnum)
            << (u_int) log2(block_size);

    return address;
}

unsigned int get_byte_offset(unsigned int address) {
```

```
    return (unsigned int) (address
            << (unsigned int) (sizeof(int) * 8 -
            log2(block_size)))
            >> (unsigned int) (sizeof(int) * 8 -
            log2(block_size));
}

unsigned char read_byte(int address, char * hit) {
    size_t temp_hit_amount = cache_memory.hit_amount;
    read_block(address >> (int) log2(block_size));
    if (cache_memory.hit_amount == temp_hit_amount+1){
        *hit = '1';
    }else{
        *hit = '0';
    }
    int slot = hit_blocknum(address); // Hay hit por el
    //read_block anterior
    return cache_memory.memory[slot].
    data[get_byte_offset((u_int) address)];
}

void write_byte_tomem(int address, unsigned char value) {
    principal_memory[address] = value;
}

char write_byte(int address, unsigned char value) {
    size_t temp_hit_amount = cache_memory.hit_amount;
    read_block(address >> (int) log2(block_size));

    int slot = hit_blocknum(address); // Hay hit por el
    //read_block anterior
    cache_memory.memory[slot].data
    [get_byte_offset((u_int) address)] = value;

    write_byte_tomem(address, value);

    if (cache_memory.hit_amount == temp_hit_amount+1){
        return '1';
    }else{
        return '0';
    }
}

int get_miss_rate() {
    size_t miss_requests = cache_memory.miss_amount * 100;
    size_t total_requests = cache_memory.miss_amount +
```

```
    cache_memory.hit_amount;
    if (total_requests == 0) return 0;
    return (int) (miss_requests / total_requests);
}

void destroy() {
    for (int i = 0; i < cache_memory.block_amount; i++)
        free(cache_memory.memory[i].data);

    free(cache_memory.memory);
}
```