




## Trabajo práctico 2: Memorias Cache

Apellido y Nombre	Padrón	Correo electrónico
Llauró, Manuel Luis	95736	llauromanuel@gmail.com
Pinto, Santiago	96850	pinto.santiago.augusto@gmail.com
Reimondo, Matias	95899	matiasreimondo@gmail.com

GitHub : <https://github.com/MatiasReimondo/Orga6620TP0>

# Índice

1. Introducción	2
2. Diseño e implementación	2
3. Modo de uso	2
4. Herramientas utilizadas y testing	2
5. Casos de prueba	3
6. Conclusión y análisis	7
7. Anexo A: Código C	8
7.1. main . . . . .	8
7.2. functions . . . . .	8

## 1. Introducción

Familiarizarse con el funcionamiento de la memoria cache implementando una simulación de una cache dada.

## 2. Diseño e implementación

Tomadas en cuenta todas las peticiones para el programa puestas en el enunciado de este trabajo, el cual se encuentra adjunto, se diseñó lo siguiente:

El archivo **main.c** llama a la función **parse\_arguments**, la cual tiene la inicialización de la cache a través de la instrucción **init()** y la lógica interna para poder leer los flags pasados al programa y a partir de estos poder ejecutar correctamente la opción pedida, utilizando las funciones creadas, correspondientes.

Los address ingresados a la cache son en sistema decimal, pero por dentro se hace un tratamiento en binario para poder conocer fácilmente el tag, index y offset del cache.

Las mayores dificultades las encontramos en detalles como el cálculo de miss y hit o la traducción de address de decimal a binario, pero fueron solo cuestiones de errores humanos pero no de concepto.

El programa fue desarrollado completamente en entornos GNU/Linux, bajo arquitecturas x86-64.

Con la suma de todo esto se logró armar el programa pedido.

## 3. Modo de uso

El ejecutable compilado no tiene dependencias con otros archivos o librerías, y puede moverse y ejecutarse desde cualquier directorio. Al ejecutarse desde una terminal sin argumentos adicionales, se le deberá pasar el programa obligatoriamente el archivo de donde obtendrá las instrucciones pedidas a la cache. Las instrucciones disponibles para la cache son:

- `read byte(address)`
- `write byte(int address, unsigned char value)`
- `get miss rate()`

## 4. Herramientas utilizadas y testing

El funcionamiento correcto del proyecto se sometió a prueba haciendo uso de varias herramientas propias de los entornos Unix-like, principalmente de *bash*, y de las *coreutils* de GNU, para armar un simple script que busque archivos de entrada en un directorio, y compare los resultados con archivos de salida. También se usa como compilador el designado por la cátedra, *gcc*.

## 5. Casos de prueba

Se realizaron para el trabajo práctico 10 casos de pruebas distintos para verificar el correcto funcionamiento del código.

Para correr las pruebas se creó un archivo `run.sh`, el cual corre todas las pruebas que se encuentran en la carpeta de Test.

A continuación se presentan las pruebas que se corren en `run.sh`. Por cuestiones de espacio, no se mostraran los archivos de texto en este informe, pero se pueden ver en el [github](#) aclarado en la carátula de este informe, en el directorio `TP22018/Tests tp2.zip` :

---

```
#!/bin/bash
gcc -Wall main.c functions.c -std=c99 -o tp2
## Algunos codigos a tener en cuenta: 139(Segmentation fault),
    255(Archivo inexistente)
#####TEST 1#####
if [ "$?" != 255 ] && [ ! -s stderr.txt ]; then ## Si stderr esta vacio
    la salida no fue dirigida correctamente, si el codigo es 255 no
    existe el file
./tp2 test1.txt > stdout.txt
DIFF=$(diff expected_result1.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test lectura en cache de un valor almacenado: ERROR";
else
    echo "Test lectura en cache de un valor almacenado: OK";
fi
####FIN TEST 1#####

#####TEST 2#####
./tp2 test2.txt > stdout.txt
DIFF=$(diff expected_result2.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test miss en un write: ERROR";
else
    echo "Test miss en un write: OK";
fi
####FIN TEST 2#####

####TEST 3#####
./tp2 test3.txt > stdout.txt
DIFF=$(diff expected_result3.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test write en cache con un hit: ERROR";
else
    echo "Test write en cache con un hit: OK";
fi
####FIN TEST 3###
```

```

####TEST 4#####
./tp2 test4.txt > stdout.txt
if [ "$?" != 253 ]; then
    echo "Test lectura fuera del rango de direcciones da error: ERROR";
else
    echo "Test lectura fuera del rango de direcciones da error: OK";
fi
####FIN TEST 4###

####TEST 5#####
./tp2 test5.txt > stdout.txt
if [ "$?" != 253 ]; then
    echo "Test escritura fuera del rango de direcciones da error: ERROR";
else
    echo "Test escritura fuera del rango de direcciones da error: OK";
fi
####FIN TEST 5###

####TEST 6#####
./tp2 prueba1.mem > stdout.txt
DIFF=$(diff expected_result6.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test prueba1.mem: ERROR";
else
    echo "Test prueba1.mem: OK";
fi
####FIN TEST 6###

####TEST 7#####
./tp2 prueba2.mem > stdout.txt
DIFF=$(diff expected_result7.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test prueba2.mem: ERROR";
else
    echo "Test prueba2.mem: OK";
fi
####FIN TEST 7###

####TEST 8#####
./tp2 prueba3.mem > stdout.txt
DIFF=$(diff expected_result8.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test prueba3.mem: ERROR";
else
    echo "Test prueba3.mem: OK";
fi

```

```
fi
####FIN TEST 8###

####TEST 9#####
./tp2 prueba4.mem > stdout.txt
DIFF=$(diff expected_result9.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test prueba4.mem: ERROR";
else
    echo "Test prueba4.mem: OK";
fi
####FIN TEST 9###

####TEST 10#####
./tp2 prueba5.mem > stdout.txt
DIFF=$(diff expected_result10.txt stdout.txt)
if [ "$DIFF" != "" ]; then
    echo "Test prueba5.mem: ERROR";
else
    echo "Test prueba5.mem: OK";
fi
####FIN TEST 10###
```

---

Los resultados y análisis instrucción por instrucción de las pruebas pedidas por enunciado fueron los siguientes:

linea		resultado		analisis
-------	--	-----------	--	----------

Prueba 1:

1		-1		Miss compulsivo (nunca estuvo alocado en cache)
2		-1		Miss compulsivo
3		-1		Miss compulsivo (el bloque 1024 se guarda en el mismo set que el adress 0, pero en la otra via)
4		-1		Miss compulsivo
5		-1		Miss compulsivo (el bloque 2050 se almacena en el set 2 en la primera via)
6		-1		Miss compulsivo (el bloque 3074 se almacena en el set 2 en la segunda via)
7		0		Hit
8		12		Hit
9		12		Hit
10		-1		Miss
11		-1		Miss
12		16		Hit
13		66		Miss Rate

Prueba 2:

1		-1		Miss compulsivo (el bloque 0 de memoria se almacena en el set 0 de cache en la primera via)
2		-1		Miss compulsivo
3		-1		Miss compulsivo
4		-1		Miss (El bloque 32 de memoria se almacena en el set 0 de cache en la segunda via)
5		0		Hit
6		20		Hit
7		-1		Miss compulsivo (EL bloque 1040 reemplaza al bloque 0 en la cache)
8		-1		Miss compulsivo (El bloque 2064 reemplaza al bloque 32 en la cache porque tambien cae en el set 0)
9		-1		Miss de conflicto (El bloque 32 habia sido reemplazado en cache y ahora vuelve reemplazando al bloque 1040)
10		20		Hit
11		70		Miss Rate

Prueba 3:

1		-1		Miss compulsivo
2		-1		Miss compulsivo
3		-1		Miss compulsivo
4		-1		Miss compulsivo
5		-1		Miss compulsivo
6		-1		Miss

```

7 | -1 | Miss
8 | -1 | Miss
9 | -1 | Miss
10 | -1 | Miss
11 | 100 | Miss Rate

```

Prueba 4:

```

1 | -1 | Miss compulsivo
2 | -1 | Miss compulsivo
3 | -1 | Miss compulsivo
4 | -1 | Miss compulsivo
5 | -1 | Miss compulsivo
6 | -1 | Miss (El bloque 0 se almacena en primera via del set 0)
7 | -1 | Miss (El bloque 1 se almacena en primera via del set 1)
8 | -1 | Miss (El bloque 2 se almacena en primera via del set 2)
9 | -1 | Miss (El bloque 3 se almacena en primera via del set 3)
10 | -1 | Miss (El bloque 4 se almacena en primera via del set 4)
11 | -1 | Miss compulsivo (El bloque 1024 se almacena en segunda via del set 0)
12 | -1 | Miss compulsivo (El bloque 2048 se almacena en primera via del set 0
    y reemplaza al bloque 0)
13 | -1 | Miss de conflicto (El bloque 0 fue reemplazado por el 2048)
14 | 2 | Hit
15 | 3 | Hit
16 | 4 | Hit
17 | 5 | Hit
18 | 76 | Miss Rate

```

Prueba 5:

```

1 | -1 | Miss compulsivo (El bloque 0 se almacena en primera via del set 0)
2 | -1 | Miss compulsivo (El bloque 1024 se almacena en segunda via del set 0)
3 | -1 | Miss compulsivo (El bloque 3072 se almacena en primera via del set 0 y
    reemplaza al bloque 0)
4 | -1 | Miss compulsivo (El bloque 2048 se almacena en segunda via del set 0 y
    reemplaza al bloque 1024)
5 | -1 | Miss de conflicto (El bloque 0 habia sido reemplazado por el 3072 y
    ahora lo reemplaza)
6 | -1 | Miss de conflicto (El bloque 3072 fue reemplazado por el 0 y ahora
    vuelve a cache pero reemplazando al bloque 2048)
7 | 100 |

```

## 6. Conclusión y análisis

Finalizada la realización del programa, se procedió con los test para comprobar y corroborar el funcionamiento correcto de la cache.

Como se puede observar en los resultados de las pruebas, muchas eran para comprobar el estado de la cache contando la cantidad de miss. Por esta razón, en todas las pruebas el miss rate fue muy alto.



## 7. Anexo A: Código C

El código fue compilado de la siguiente manera:  
`gcc -Wall -s functions.c main.c -std=c99 -o tp2`

### 7.1. main

---

```
#include "functions.h"

int main(int argc, char * argv[]) {
    return parse_arguments(argc,argv);
}
```

---

### 7.2. functions

---

```
#ifndef CACHE_FUNCTIONS_H
#define CACHE_FUNCTIONS_H

#define ERROR_NO_FILE -10

static const char DELIMITERS[] = " ,\n";

//Parsea el archivo y hace las correspondientes llamadas a las primitivas
int parse_arguments(int argc, char * argv[]);

//Realiza el malloc de las variables globales
void pre_run();

void init();

unsigned char read_byte(int address);

int write_byte(int address, unsigned char value);

unsigned int get_miss_rate();

void free_cache();

int ilog2 (int x);

int extract_offset (int address, int sets, int block_size);

#endif //CACHE_FUNCTIONS_H
```

---

```
#include "functions.h"
#define _GNU_SOURCE
#include <stdio.h>
#include <errno.h>
#include <string.h>
```

---

```

#include <stdlib.h>

#define SIZE_OF_MEMORY 4096
#define SIZE_OF_BLOCK 32
#define BLOCK_NUMBER 16
#define NO_TAG -2
#define SETS_CACHE 2
#define MISS_SIGNAL -1
#define HIT_SIGNAL 0
#define OUT_OF_BOUNDS -3
#define BAD_ARGS -4
#define ZERO 0
// Estructura de un bloque cache
// data: son los 32 bytes del bloque
// use last: se va cambiando a medida de que se accede en la lectura 1 o
// 0 dependiendo de a cual se acceda
struct cache_block{
    unsigned char * data;
    int bit_dirty;
    int tag;
    int use_last;
};

//Memoria principal
unsigned char * memory;

//Cantidad de miss
unsigned int miss;

//Cantidad de accesos a memoria
unsigned int access;

// Via 0 de la cache
struct cache_block v0[BLOCK_NUMBER];

// Via 1 de la cache
struct cache_block v1[BLOCK_NUMBER];

// Iniciacion de la metadata de la cache y los malloc
void pre_run(){
    memory = malloc(SIZE_OF_MEMORY);
    for (int i = 0; i <BLOCK_NUMBER ; ++i) {
        v0[i].data = malloc(SIZE_OF_BLOCK);
        v0[i].bit_dirty = 0;
        v0[i].tag = NO_TAG;
        v0[i].use_last = 0;
        v1[i].data = malloc(SIZE_OF_BLOCK);
        v1[i].bit_dirty = 0;
        v1[i].tag = NO_TAG;
        v1[i].use_last = 0;
    }
}

```

```

//Init pedido por el enunciado, lo del bit de dirty lo agregue, es una
    de als dudas
void init(){
    miss = 0;
    access = 0;
    for (int i = 0; i < BLOCK_NUMBER ; ++i) {
        memset(v0[i].data, '\0', SIZE_OF_BLOCK);
        memset(v1[i].data, '\0', SIZE_OF_BLOCK);
        v0[i].bit_dirty = 1;
        v1[i].bit_dirty = 1;
    }
}

}

unsigned char read_byte(int address){
    int bloque_cache = address%BLOCK_NUMBER;
    int tag = address/BLOCK_NUMBER;
    int offset = extract_offset(address,SETS_CACHE,SIZE_OF_BLOCK);
    access = access+ 1;

    if((v0[bloque_cache].tag == tag) && (v0[bloque_cache].bit_dirty!=
        1)){
        v0[bloque_cache].use_last = 1;           //Si encuentra en la
            cache el dato, setea esa via como la ultima usada
        v1[bloque_cache].use_last = 0;           // y la otra como no
        printf("%u \n",v0[bloque_cache].data[offset]);
        return v0[bloque_cache].data[offset];

    }else if((v1[bloque_cache].tag == tag) &&
        (v1[bloque_cache].bit_dirty!= 1)){
        v0[bloque_cache].use_last = 0;
        v1[bloque_cache].use_last = 1;
        printf("%u \n",v1[bloque_cache].data[offset]);
        return v1[bloque_cache].data[offset];
    }else{
        miss = miss +1;                           //Sino encuentra el dato
            lo guarda en la que fue menor usada
        printf("%d \n", MISS_SIGNAL);
        if((v0[bloque_cache].bit_dirty!=1)
            &&(v1[bloque_cache].bit_dirty!=1)){
            if(v0[bloque_cache].use_last){
                v1[bloque_cache].data[offset] = memory[address];
                v1[bloque_cache].bit_dirty = 0;
                v1[bloque_cache].tag = tag;
                v1[bloque_cache].use_last = 1;
                v0[bloque_cache].use_last = 0;
            }else {
                v0[bloque_cache].data[offset] = memory[address];
                v0[bloque_cache].bit_dirty = 0;
                v0[bloque_cache].tag = tag;
                v0[bloque_cache].use_last = 1;
                v1[bloque_cache].use_last = 0;
            }
        }
    }
}

```

```

        }else if(v0[bloque_cache].bit_dirty==1){
            v0[bloque_cache].data[offset] = memory[address];
            v0[bloque_cache].bit_dirty = 0;
            v0[bloque_cache].tag = tag;
            v0[bloque_cache].use_last = 1;
            v1[bloque_cache].use_last = 0;
        }else{
            v1[bloque_cache].data[offset] = memory[address];
            v1[bloque_cache].bit_dirty = 0;
            v1[bloque_cache].tag = tag;
            v1[bloque_cache].use_last = 1;
            v0[bloque_cache].use_last = 0;
        }
        return MISS_SIGNAL;
    }
}

int write_byte(int address, unsigned char value){
    int bloque_cache = address%BLOCK_NUMBER;
    int tag = address/BLOCK_NUMBER;
    int offset = extract_offset(address,SETS_CACHE,SIZE_OF_BLOCK);
    access = access+ 1;

    if(v0[bloque_cache].tag == tag){
        v0[bloque_cache].data[offset] = value;
        v0[bloque_cache].bit_dirty = 0;
        memory[address] = value;
        printf("%d \n",HIT_SIGNAL);
        return 0;
    }else if(v1[bloque_cache].tag == tag){
        v1[bloque_cache].data[offset] = value;
        v1[bloque_cache].bit_dirty= 0;
        memory[address] = value;
        printf("%d \n",HIT_SIGNAL);
        return 0;
    }else{
        memory[address] = value;
        miss = miss +1;
        printf("%d \n",MISS_SIGNAL);
        return -1;
    }
}

}

unsigned int get_miss_rate(){
    unsigned int missRate = (miss*100)/access;
    printf("%d \n",missRate);
    return missRate;
}

int parse_arguments(int argc, char * argv[]){
    pre_run();
    init();
    FILE *fi;

```

```

int exe_code = 0;
size_t len = 32;
char *line = NULL;
char *argument1 = NULL;
char *argument2 = NULL;
char *argument3 = NULL;
if(argc !=2){
    printf("Ingrese un nombre valido de archivo \n");
    return BAD_ARGS;
}else{
    fi = fopen(argv[1],"r");
    if(errno != 0){
        fprintf(stderr,"No se pudo abrir el archivo \n");
        exe_code = ERROR_NO_FILE;
        return exe_code;
    }
    while(getline(&line,&len,fi)!=-1){
        argument1 = strtok(line,DELIMITERS);
        if(strcmp("W",argument1) == 0){
            argument2 = strtok(NULL,DELIMITERS);
            int arg2 = atoi(argument2);
            if(arg2 > SIZE_OF_MEMORY || arg2 < ZERO){
                printf("Direccion de memoria incorrecta \n");
                return OUT_OF_BOUNDS;
            }
            argument3 = strtok(NULL,DELIMITERS);
            unsigned char arg3 = (unsigned char) atoi(argument3);
            write_byte(arg2,arg3);
        }
        if(strcmp("R",argument1) == 0){
            argument2 = strtok(NULL,DELIMITERS);
            int arg2 = atoi(argument2);
            if(arg2 > SIZE_OF_MEMORY || arg2 < ZERO){
                printf("Direccion de memoria incorrecta \n");
                return OUT_OF_BOUNDS;
            }
            read_byte(arg2);
        }
        if(strcmp("MR",argument1) == 0){
            get_miss_rate();
        }
    }
    free_cache();
    return 0;
}

}

void free_cache(){
    for (int i = 0; i <BLOCK_NUMBER ; ++i) {
        free(v0[i].data);
        free(v1[i].data);
    }
}

```

```

    }
}

//Usamos operadores de bits para ,segun el address , obtener el offsetr

int ilog2 (int x){
    int result = 0;

    while (x != 0) {
        result++;
        x = x >> 1;
    }
    return result;
}

int extract_offset (int address, int sets, int block_size){

    int offset_bits = ilog2(block_size);

    int offset = address & ((1 << offset_bits) - 1);
    return offset;
}

```

---