66.20 - Organización de Computadoras
Trabajo Práctico 2 - Memorias Caché

Andrew Parlane

17 de mayo de 2018

Índice

1. Resumen

El objetivo de este trabajo se basa en aplicar los conocimientos aprendidos en clase acerca caches. Para eso desarrollé un programa en C que simula un caché.

2. Introducción

El programa que desarrollé lee comandos desde un archivo texto y simula lecturas y escrituras a memoria usando un cache *Write Through / No Write Allocate*. Hay tres comandos soportado *R* para *Read*, *W* para *Write* y *MR* para *Miss Rate*. La memoria principal es 4KB, y el cache es 1KB asociativa por conjuntos con dos vías. El tamaño de bloque es 32 bytes.

3. Compilacion y ejecución del programa

Esta sección describe los mecanismos para la compilación y ejecución del programa.
Para compilar usando el Makefile:

- Estando parados en la carpeta donde se encuentran los archivos fuente, ejecutamos el siguiente comando:

  ```
  make
  ```

Para ejecutar:

- Estando parados en la carpeta donde se encuentran el archivo ejecutable, corremos para cada archivo lo siguiente:

  ```
  ./tp2 [options] file
  ```

- Si se quiere ver el help, que especifica las opciones disponibles al momento de invocar el programa:

  ```
  ./tp2 -h
  ```

- Si se quiere ver la versión del programa:

  ```
  ./tp2 -V
  ```

4. Implementación

4.1. Implementación en C

Usé algunas definiciones y estructuras para simplificar la implementación de memoria principal y el caché:

- *Bloque* - El bloque es un struct que contiene un array de TAMANO_DE_BLOQUE bytes.
- *LineaDeCache* - Una línea de caché contiene un Bloque de data, el TAG y un flag válido.
- *ViaDeCache* - Una vía de caché contiene CONJUNTOS_EN_CACHE líneas.
- *Cache* - El cache contiene NUMERO_DE_VIAS vías y un array de CONJUNTOS_EN_CACHE de tipo bool que uso por el algoritmo LRU.

A continuación enumero las funciones definidas en el programa que se usarán luego de la verificación y validación de los parámetros de entrada:

- **init**
  parámetros:

  descripción: función que inicializa el cache, escribiendo cero al bit válido de todos las líneas del cache.

- **dumpCache**
  parámetros:

  descripción: función que escribe el contenido del cache a stdout. Solo usado si DEBUG es definido.

- **decodeAddress**
  parámetros: uint32_t addr
  uint32_t *tag
  uint32_t *indice
  uint32_t *via
  uint32_t *offset
  uint32_t *mpBloque

  descripción: función que convierta una dirección de memoria principal hasta el índice de conjunto, el TAG, el número de bloque en MP y el offset del byte en ese bloque. Después busca en los dos vías del cache a ver si el byte querido es en el caché. Si no es encontrado incrementa el contador de misses.

- **read_byte**
  parámetros: int address

  descripción: función que lee el byte a la dirección: *address*. Si no es en el caché copia el bloque querido desde MP hasta el caché, reemplazando la línea usado más lejos en el pasado.

- **write_byte**
  parámetros: int address
  int value

  descripción: función que escribe el byte: *value* a la dirección: *address* en MP. Si es en caché actualiza el valor en el cache también.

- **get_miss_rate**
  parámetros:

  descripción: función que devuelva el *Miss Rate* como un porcentaje redondeado al entero más próximo. Si no había accesos al caché devuelva 0 %.

- **usage**
  parámetros: FILE *stream
  const char *nuestroNombre

  descripción: función que muestra el help de la aplicación.

- **skipLine**
  parámetros: FILE *stream

  descripción: función que bota todos los caracteres en el archivo hasta el final de la línea corriente. Uso esto cuando había un error en un comando.

- **leerLongLong**
  parámetros: FILE *f
  long long *ll
  bool *OK
  bool *eof
  bool *newLine

  descripción: función que lee un entero de desde el archivo. Lo copié desde TP1.

- **leerComando**
  parámetros: FILE *f
  bool *eof
  Comando *cmd

  descripción: función que lee el comandos siguiente desde el archivo.

- **leerArchivo**
  parámetros: const char *archivo

  descripción: función que lee el archivo, y actúa sobre los comandos encontrados.

5. Pruebas

Realicé las pruebas incluido con el enunciado.
- **prueba1.mem**
- **prueba2.mem**
- **prueba3.mem**

```
$ ./tp2 pruebas/prueba1.mem
Miss Rate: 67%

$ ./tp2 pruebas/prueba2.mem
Miss Rate: 50%

$ ./tp2 pruebas/prueba3.mem
Miss Rate: 60%

$ ./tp2 pruebas/prueba4.mem
Miss Rate: 53%

$ ./tp2 pruebas/prueba5.mem
Miss Rate: 100%
```

También He realizado unas pruebas extras con archivos mal formados.

6. Conclusiones

El desarrollo de este trabajo me permitió llevar a la práctica los conocimientos adquiridos acerca de un caché asociativa por conjuntos WT/nWA.

7. Código

```
#include <stdio.h>
#include <stdlib.h>
#include <libgen.h>
#include <getopt.h>
#include <ctype.h>
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include <errno.h>
#include <string.h>

#define MAJOR_VERSION   1
#define MINOR_VERSION   1
```

```
//#define DEBUG

// No pueden cambiar estos sin cambiando el código
#define TAMANO_DE_BLOQUE    (32)
#define TAMANO_DE_MEMORIA   (4096)
#define TAMANO_DE_CACHE     (1024)
#define NUMERO_DE_VIAS      (2)

// Valores calculados
#define BLOQUES_EN_MP       ((TAMANO_DE_MEMORIA)/(TAMANO_DE_BLOQUE))
#define LINEAS_EN_CACHE     ((TAMANO_DE_CACHE)/(TAMANO_DE_BLOQUE))
#define CONJUNTOS_EN_CACHE  ((LINEAS_EN_CACHE)/(NUMERO_DE_VIAS))

typedef enum
{
    Comando_R = 0,
    Comando_W,
    Comando_MR,
} Comando;

typedef struct
{
    uint8_t data[TAMANO_DE_BLOQUE];
} Bloque;

typedef struct
{
    Bloque data;

    uint32_t tag;
    bool valido;
} LineaDeCache;

typedef struct
{
    LineaDeCache linea[CONJUNTOS_EN_CACHE];
} ViaDeCache;

typedef struct
{
    ViaDeCache via[NUMERO_DE_VIAS];

    bool via1estuvoUsadoUltimo[CONJUNTOS_EN_CACHE];
} Cache;

static const struct option _gLongOptions[] =
{
    {"help",    no_argument,        0, 'h' },
    {"version", no_argument,        0, 'V' },
    {0,         0,                  0,  0  }
};

static Bloque   _gMemoriaPrincipal[BLOQUES_EN_MP];
static Cache    _gCache;

static uint32_t _gAccessosTotal;
static uint32_t _gMisses;
```

```
// ---------------------------------------------------------------------
// Funciones obligatorio por el TP
// no cambias los nombres, argumentos
// o tipo de resultados
// ---------------------------------------------------------------------

static void init()
{
    _gAccessosTotal = 0;
    _gMisses = 0;

    uint32_t via;
    for (via = 0; via < NUMERO_DE_VIAS; via++)
    {
        uint32_t linea;
        for (linea = 0; linea < CONJUNTOS_EN_CACHE; linea++)
        {
            _gCache.via[via].linea[linea].valido = false;
        }
    }
}

#ifdef DEBUG
static void dumpCache()
{
    printf("IDX    VIA1    VIA2    Ultimo\n");
    int i;
    for (i = 0; i < (CONJUNTOS_EN_CACHE); i++)
    {
        printf("%3X    ", i);
        if (_gCache.via[0].linea[i].valido)
        {
            printf("%4X    ", _gCache.via[0].linea[i].tag);
        }
        else
        {
            printf("INVA    ");
        }
        if (_gCache.via[1].linea[i].valido)
        {
            printf("%4X    ", _gCache.via[1].linea[i].tag);
        }
        else
        {
            printf("INVA    ");
        }
        printf("%u\n", _gCache.via1estuvoUsadoUltimo[i]);
    }
    printf("\n");
}

#endif

static bool decodeAddress(uint32_t addr, uint32_t *tag, uint32_t *indice, uint32_t *via, uint32_t *off
{
    // Hay 32 bytes cada bloque, así offset = 5 bits
    // Hay 16 conjuntos, así indice = 4 bits
    // Y TAG es los demás
```

```
    // |---MPBLOQUE----|--OFFSET--| <-- MP
    // |---TAG---|-IDX-|--OFFSET--| <-- Cache
    //  15      9 8   5 4        0

    *offset = addr & 0x1F;
    *indice = (addr >> 5) & 0x0F;
    *tag = (addr >> 9);
    *mpBloque = (addr >> 5);

#ifdef DEBUG
    printf("addr %X:\n", addr);
    printf("  tag: %X\n", *tag);
    printf("  idx: %X\n", *indice);
    printf("  off: %X\n", *offset);
    printf("  mp:  %X\n", *mpBloque);
#endif

    // Comproba si es un HIT
    LineaDeCache *linea0 = &_gCache.via[0].linea[*indice];
    LineaDeCache *linea1 = &_gCache.via[1].linea[*indice];

    if (linea0->valido &&
        linea0->tag == *tag)
    {
        // HIT en via 0
        *via = 0;
#ifdef DEBUG
        printf("  HIT\n\n");
#endif
        return true;
    }

    if (linea1->valido &&
        linea1->tag == *tag)
    {
        // HIT en via 0
        *via = 1;
#ifdef DEBUG
        printf("  HIT\n\n");
#endif
        return true;
    }

    // MISS
#ifdef DEBUG
    printf("  MISS\n\n");
#endif
    _gMisses++;
    return false;
}

static int read_byte(int address)
{
    _gAccessosTotal++;

    uint32_t indice;
    uint32_t tag;
    uint32_t via;
```

```
    uint32_t offset;
    uint32_t mpBloque;

    unsigned char data;
    bool hit = decodeAddress(address, &tag, &indice, &via, &offset, &mpBloque);
    if (hit)
    {
        // hit
    }
    else
    {
        // miss

        // si hay un linea que no es válido reemplazamos eso
        if (!_gCache.via[0].linea[indice].valido)
        {
            via = 0;
        }
        else if (!_gCache.via[1].linea[indice].valido)
        {
            via = 1;
        }
        else
        {
            // LRU
            via = !_gCache.via1estuvoUsadoUltimo[indice];

            // Cache es WT así no puede ser dirty
        }

        memcpy(_gCache.via[via].linea[indice].data.data,
               _gMemoriaPrincipal[mpBloque].data,
               TAMANO_DE_BLOQUE);

        _gCache.via[via].linea[indice].tag = tag;
        _gCache.via[via].linea[indice].valido = true;
    }

    // leer Data de cache
    data = _gCache.via[via].linea[indice].data.data[offset];

#ifdef DEBUG
    printf("Read byte %u\n", data);
#endif

    // actualiza ultimo uso
    _gCache.via1estuvoUsadoUltimo[indice] = via;

    return hit ? data : -1;
}

static int write_byte(int address, unsigned char value)
{
    _gAccessosTotal++;
    uint32_t indice;
    uint32_t tag;
    uint32_t via;
    uint32_t offset;
```

```
    uint32_t mpBloque;

    bool hit = decodeAddress(address, &tag, &indice, &via, &offset, &mpBloque);
    if (hit)
    {
        // hit
        // actualizar el cache
        _gCache.via[via].linea[indice].data.data[offset] = value;

        // actualiza ultimo uso
        _gCache.via1estuvoUsadoUltimo[indice] = via;
    }
    else
    {
        // miss
        // ~WA así no hacemos nada
    }

    // actualizar MP (estamos WT)
    _gMemoriaPrincipal[mpBloque].data[offset] = value;

    return hit ? 0 : -1;
}

static unsigned int get_miss_rate()
{
    if (_gAccessosTotal == 0)
    {
        // _gMisses / 0
        // _gMisses debería estar 0 también
        // así decimos 0%
        return 0;
    }
    else
    {
        double porcentaje = ((double)_gMisses * 100.0l) / (double)_gAccessosTotal;
        return (unsigned int)round(porcentaje);
    }
}

// ------------------------------------------------------------------
// Final de funciones obligatorio por el TP
// ------------------------------------------------------------------

static void usage(FILE *stream, const char *nuestroNombre)
{
    fprintf(stream,
           "Usage:\n"
           "  %s -h\n"
           "  %s -V\n"
           "  %s filename\n"
           "Options:\n"
           "  -h, --help Prints usage information.\n"
           "  -V, --version Prints version information.\n"
           "\n\n",
           nuestroNombre, nuestroNombre, nuestroNombre);
    // necesitamos usar nuestroNombre, nuestroNombre, nuestroNombre
    // porque no soportamos %1$s
```

```
}

static void skipLine(FILE *f)
{
    while (1)
    {
        int res = fgetc(f);
        if (res == EOF ||
            (char)res == '\n')
        {
            return;
        }
    }
}

// lea carácter por carácter deshaciendo whitespace
// hasta encontrar [0-9-]. Después comenzar a leer números
// [0-9]. hasta obtenemos EOF, \n, \r, ' ', \t, ','
// Es un error si encontramos algún otro carácter.
// devolver no 0 si hay un error
// *OK = 1 -> hay un integer valido en *ll
// *eof = 1 -> no hay más a leer
// *newLine = 1 -> encontramos nueva línea
static bool leerLongLong(FILE *f, long long *ll, bool *OK, bool *eof, bool *newLine)
{
    *OK = false;
    *eof = false;
    *newLine = false;

    // soportamos signed 64 bits:
    // máx = 0x7FFF_FFFF_FFFF_FFFF =  9223372036854775807
    // min = 0x8000_0000_0000_0000 = -9223372036854775808
    // así max input legal es 20 cáracters +1 por NULL terminator
#define MAX_CHARS 20
    char buff[MAX_CHARS + 1];
    uint idx = 0;

    bool comenzandoLeerInt = false;
    while (1)
    {
        int res = fgetc(f);
        if (res == EOF)
        {
            *eof = true;
            // si tenemos algo en buff, convertimos ahora
            if (*OK)
            {
                buff[idx] = '\0';
                *ll = strtoll(buff, NULL, 10);
                if (errno != 0)
                {
                    fprintf(stderr, "Failed to convert %s to long long, error: %s\n", buff, strerror(e
                    return false;
                }
                return true;
            }
            else if (comenzandoLeerInt)
            {
```

```
            // solo podríamos estar aquí si leemos
            // un '-' y después nada, eso es un error
            fprintf(stderr, "Found invalid entry \"-\"\n");
            return false;
        }
        else
        {
            // eof but no error
            return true;
        }
    }

    char c = (char)res;
    if (c == '\r' || c == '\n')
    {
        *newLine = true;
    }

    if (!comenzandoLeerInt)
    {
        // Todavía no cemenzamos a leer el int
        if (c == ' ' || c == '\t')
        {
            // ignoramos
            continue;
        }
        else if (*newLine)
        {
            // nuevo línea, terminamos.
            return true;
        }
        else if (c >= '0' && c <= '9')
        {
            // válido
            buff[idx++] = c;
            comenzandoLeerInt = true;
            *OK = true;
        }
        else if (c == '-')
        {
            // también válido pero el int todavía no es OK
            // porque necesitamos un número después de un -
            buff[idx++] = c;
            comenzandoLeerInt = true;
        }
        else
        {
            // error
            fprintf(stderr, "Found invalid character %c\n", c);
            return false;
        }
    }
    else
    {
        // ya estamos leyendo data
        if (c == ' ' || c == '\t' ||
            c == ',' || *newLine)
        {
```

```
                // terminamos
                if (*OK)
                {
                    buff[idx] = '\0';
                    *ll = strtoll(buff, NULL, 10);
                    if (errno != 0)
                    {
                        fprintf(stderr, "Failed to convert %s to long long, error: %s\n", buff, strerr
                        return false;
                    }
                    return true;
                }
                else
                {
                    // solo podríamos estar aquí si leemos
                    // un '-' y después nada, eso es un error
                    fprintf(stderr, "Found invalid entry \"-\"\n");
                    return false;
                }
            }
            else if (c == '-')
            {
                // un - aquí no es válido porque estámos en el medio
                // de un int.
                fprintf(stderr, "Found \"-\" in the middle of an integer\n");
                return false;
            }
            else if (c >= '0' && c <= '9')
            {
                // válido
                if (idx >= MAX_CHARS)
                {
                    fprintf(stderr, "Integer read was too large to fit into a long long\n");
                    return false;
                }
                buff[idx++] = c;
                *OK = true;
            }
            else
            {
                // error
                fprintf(stderr, "Found invalid character %c\n", c);
                return false;
            }
        }
    }
}

static bool leerComando(FILE *f, bool *eof, Comando *cmd)
{
    // comando más largo es MR, así usamos un buffer de
    // 3 bytes con [2] = NULL
    char buff[3];
    memset(buff, 0, sizeof(buff));

    // read char 1, puede ser M, R o W
    // ignoramos espacio
    while (1)
```

```
{
    int res = fgetc(f);
    if (res == EOF)
    {
        *eof = true;
        return false;
    }

    char c = (char)res;
    if (isspace(c))
    {
        continue;
    }

    // es un carácter no espacio
    // es M, R o W?
    if (c != 'M' &&
        c != 'W' &&
        c != 'R')
    {
        // invalido
        fprintf(stderr, "Read start of invalid command, skipping line.\n");

        // leer hasta '\n'
        skipLine(f);

        return false;
    }

    // es válido
    buff[0] = c;
    break;
}

// ahora leer el comando hasta encontramos un espacio
int idx = 1;
while (1)
{
    int res = fgetc(f);
    if (res == EOF)
    {
        *eof = true;
    }
    if (isspace((char)res) ||
        *eof)
    {
        // comando es válido?
        if (strcmp(buff, "R") == 0)
        {
            *cmd = Comando_R;
        }
        else if (strcmp(buff, "W") == 0)
        {
            *cmd = Comando_W;
        }
        else if (strcmp(buff, "MR") == 0)
        {
            *cmd = Comando_MR;
```

```
            }
            else
            {
                // invalido
                fprintf(stderr, "Read start of invalid command, skipping line.\n");

                // leer hasta '\n'
                skipLine(f);

                return false;
            }

            return true;
        }

        if (idx > 1)
        {
            fprintf(stderr, "Command too long\n");
            // leer hasta '\n'
            skipLine(f);

            return false;
        }

        buff[idx++] = (char)res;
    }
}

static int leerArchivo(const char *archivo)
{
    FILE *f = fopen(archivo, "r");
    if (f == NULL)
    {
        fprintf(stderr, "%s: No such file or directory\n", archivo);
        return 1;
    }

#ifdef DEBUG
    dumpCache();
#endif


    while (1)
    {
        Comando cmd;
        bool eof = false;
        if (!leerComando(f, &eof, &cmd))
        {
            if (eof)
            {
                break;
            }
        }
        else
        {
            if (cmd == Comando_MR)
            {
                uint32_t mr = get_miss_rate();
```

```
            printf("Miss Rate: %u%%\n", mr);
        }
        else
        {
            long long direccion;
            bool OK;
            bool newLine;
            if (!leerLongLong(f, &direccion, &OK, &eof, &newLine))
            {
                if (eof)
                {
                    fprintf(stderr, "EOF in middle of command");
                    break;
                }
                fprintf(stderr, "Failed to read address\n");
                if (!newLine)
                {
                    skipLine(f);
                }
                continue;
            }
            else if ((direccion < 0) ||
                     (direccion >= (TAMANO_DE_MEMORIA)))
            {
                fprintf(stderr, "Address out of bounds\n");
                if (!newLine)
                {
                    skipLine(f);
                }
                continue;
            }

            if (cmd == Comando_R)
            {
#ifdef DEBUG
                printf("R %u\n", direccion);
#endif
                read_byte((int)direccion);
            }
            else
            {
                if (newLine)
                {
                    fprintf(stderr, "Comand ended too early");
                    continue;
                }

                long long valor;
                if (!leerLongLong(f, &valor, &OK, &eof, &newLine))
                {
                    if (eof)
                    {
                        fprintf(stderr, "EOF in middle of command");
                        break;
                    }
                    fprintf(stderr, "Failed to read value\n");
                    if (!newLine)
                    {
```

```
                        skipLine(f);
                    }
                    continue;
                }
                else if ((valor < 0) ||
                         (valor > 0xFF))
                {
                    fprintf(stderr, "Value out of bounds\n");
                    if (!newLine)
                    {
                        skipLine(f);
                    }
                    continue;
                }

#ifdef DEBUG
                printf("W %u, %u\n", direccion, valor);
#endif
                write_byte((int)direccion, (uint8_t)valor);
            }
        }
    }
#ifdef DEBUG
    dumpCache();
#endif
}

#ifdef DEBUG
    dumpCache();
#endif

    fclose(f);

    return 0;
}

int main(int argc, char **argv)
{
    // usamos argv[0] como el nombre del aplicación
    // pero solo queremos el archivo, no la ruta
    const char *nuestroNombre = basename(argv[0]);

    // clear errors
    opterr = 0;

    // parse short options
    while (1)
    {
        // obtener el siguiente argumento
        int option_index = 0;
        int c = getopt_long(argc, argv, "hV", _gLongOptions, &option_index);

        if (c == -1)
        {
            // no hay más
            break;
        }
```

```
        switch (c)
        {
            case 'h':
            {
                usage(stdout, nuestroNombre);
                // no seguimos despues de -h
                return 0;
            }
            case 'V':
            {
                printf("%s: Version %u.%u\n", nuestroNombre, MAJOR_VERSION, MINOR_VERSION);
                // no seguimos despues de -V
                return 0;
            }
            case '?':
            {
                if (optopt == 'o')
                {
                    fprintf(stderr, "Option '-%c' requires an argument.\n\n", optopt);
                }
                else if (isprint(optopt))
                {
                    // es un argumento, pero no es uno que esperamos
                    fprintf (stderr, "Unknown option '-%c'.\n\n", optopt);
                }
                else
                {
                    // solo muestra el usage
                }
                usage(stderr, nuestroNombre);
                return 1;
            }
            default:
            {
                usage(stderr, nuestroNombre);
                return 1;
            }
        }
    }

    if (optind == argc)
    {
        fprintf(stderr, "filename is required\n\n");
        usage(stderr, nuestroNombre);
        return 1;
    }

    if ((optind + 1) != argc)
    {
        fprintf(stderr, "Too many arguments\n\n");
        usage(stderr, nuestroNombre);
        return 1;
    }

    init();
    return leerArchivo(argv[optind]);
}
```