Trabajo Práctico N^0 0 "Infraestructura básica"

Tomás Botalla, *Padrón 96356*
tbotalla@gmail.com
Samanta Loiza, *Padrón 91935*
samiloiza@gmail.com
Nahuel Sosa, *Padrón 919289*
nahuelmartin.sosa@yahoo.com.ar

1er. Cuatrimestre de 2016
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Índice

Resumen

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa y su correspondiente documentación que resuelvan el problema descripto más abajo.

1. Introducción

2. Compilación y Ejecución del programa

Nota: Para poder compilar el programa debemos enviar los archivos fuente a NetBSD por medio del túnel creado con ssh.

Para compilar:

- Parados en la carpeta donde tenemos los fuente ejecutamos.

```
gcc -Wall -g -std=c99 tp0.c tp0.h -o tp0
```

o bien escribiendo el comando

```
make
```

Para ejecutar:

- Parados en la carpeta donde tenemos el ejecutable generado por la compilación, el programa se invoca a través de línea de comandos de la siguiente manera:

```
 cat fileTest/test1.txt | ./tp0
cat fileTest/test1.txt | ./tp0 >out1.txt
./tp0 <test1.txt
./tp0 <test1.txt>out1.txt
```

El menú de ayuda especifica las opciones disponibles al momento de invocar el programa:

```
Usage:
./tp0 -h
./tp0 -V
OPTIONS:
   -h, --help Print this information and quit.
   -V, --version Print version and quit
```

Ejemplo:

```
   ./tp0 <in.txt >out.txt
    cat in.txt | ./tp0 >out.txt
```

Para generar el código assembly generado por el compilador de MIPS32

```
gcc -Wall -std=c99 -O0 -S -mrnames tp0.c
```

3. Implementación

3.1. Desarrollo

3.2. Implementación en C

Implementamos diversas funciones para este trabajo, las cuales se encargan de:

- Imprimir ayuda:

  void printHelp()

- Checkear los argumentos recibidos:

  int checkArguments(int,char[])*

- Crear matriz:

  matrix_t create_matrix(size_t rows, size_t cols)

- Destruir matriz:

  void destroy_matrix(matrix_t m)

- Obtener la dimensión de la matriz:

  int readMatrixDimension()

- Leer matriz:

  matrix_t readMatrix(matrix_t matrix)

- Imprimir matriz:

  int print_matrix(matrix_t m)

- Multiplicar dos matrices:

  matrix_t matrix_multiply(matrix_t m1, matrix_t m2)

4. Pruebas

Se utilizaron los siguientes archivos de prueba

- *badDim.txt*: `a 1.0 2.0 3.0 4.0 1.0 2.0 2.2 4.0`

- *badCant.txt*: `3 10 1.2 3.5`

- *charTest.txt*
  ```
  2 1.0 2.0 3.0 4.0 1.0 2.0 a 4.0
  ```

- *emptyfile.txt*

- *test1.txt*
  ```
  2 1.0 2.0 3.0 4.0 1.0 2.0 3.0 4.0
  3 1.0 2.0 3.0 4.0 5.0 6.1 3.0 2.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0
  0.0 0.0 1.0
  ```

- *test2.txt*
  ```
  3 -1 3.5 2.3 6.9 -9.9 2.3 1 0 10.3 8.12 1 12 13 4.2 15.2 1.6 9.3
  6.3
  ```

- *test3.txt*
  ```
  2 -1.3 2.5 -9.3 12 3.2 7.4 -2 8
  3 -6.2 2.0 3.0 4.0 2.3 6.1 3.0 -7.4 1.0 9.12 0.0 -1 0.0 1.0 0.0
  0.0 0.0 1.0
  ```

- *test4.txt*
  ```
  2 0.000000 -3.000000 -9.000000 11.000000 -1.000000 -4.000000 -1.000000
  -4.000000
  ```

- *testHigh.txt*
  ```
  4 5.200 13.900 4.200 1.500 1.500 6.200 14.700 0.800 17.200 6.300
  1.800 0.900 0.600 3.400 5.300 4.700 6.300 1.400 17.300 12.400
  11.300 0.800 1.400 3.700 5.900 3.400 16.300 9.400 6.500 12.300
  11.700 18.200
  3 12.300 18.100 -0.900 -6.500 12.700 11.300 6.400 3.200 7.400
  4.700 3.400 11.300 16.400 -8.300 9.100 15.700 -17.100 4.600
  ```

4.1. 1^o Prueba

En esta prueba probamos que la matriz sea válida.

```
cat fileTest/charTest.txt | ./tp0
      Matriz inválida
```

4.2. 2^o Prueba

```
cat fileTest/emptyfile.txt | ./tp0
  Error: Matriz inválida para multiplicación
```

4.3. 3^o Prueba

```
cat fileTest/badDim.txt | ./tp0
   Error: Matriz inválida para multiplicación
```

4.4. 4^o Prueba

```
cat fileTest/badCant.txt | ./tp0
   Error: Matriz inválida
```

4.5. 5^o Prueba

```
./tp0 <fileTest/test1.txt
   2 7.000000 10.000000 15.000000 22.000000
   3 1.000000 2.000000 3.000000 4.000000 5.000000 6.100000 3.000000
2.000000 1.000000
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido. Si queremos redireccionar el resultado de salida `./tp0 <fileTest/test1.txt >fileTest/out.txt`

- *out.txt*:

```
2 7.000000 10.000000 15.000000 22.000000
3 1.000000 2.000000 3.000000 4.000000 5.000000 6.100000 3.000000
2.000000 1.000000
```

4.6. 6^o Prueba

```
./tp0 <fileTest/test2.txt
   3 41.060000 35.090000 55.690000 -68.992000 -13.290000 -53.190000
24.600000 96.790000 76.890000
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido. Si queremos redireccionar el resultado de salida `./tp0 <fileTest/test2.txt >fileTest/out.txt`

- *out.txt*:

```
3 41.060000 35.090000 55.690000 -68.992000 -13.290000 -53.190000
24.600000 96.790000 76.890000
```

4.7. 7^o Prueba

```
./tp0 <fileTest/test3.txt
   2 -9.160000 10.380000 -53.760000 27.180000
   3 -56.544000 2.000000 9.200000 36.480000 2.300000 2.100000 27.360000
-7.400000 -2.000000
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido. Si queremos redireccionar el resultado de salida `./tp0 <fileTest/test3.txt >fileTest/out.txt`

- *out.txt*:

```
2 -9.160000 10.380000 -53.760000 27.180000
3 -56.544000 2.000000 9.200000 36.480000 2.300000 2.100000 27.360000
-7.400000 -2.000000
```

4.8. 8^o Prueba

```
./tp0 <fileTest/test4.txt
   2 3.000000 12.000000 -2.000000 -8.000000
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido. Si queremos redireccionar el resultado de salida `./tp0 <fileTest/test4.txt >fileTest/out.txt`

- *out.txt*:

```
2 3.000000 12.000000 -2.000000 -8.000000
```

4.9. 9^o Prueba

```
./tp0 <fileTest/testHigh.txt
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido. Si queremos redireccionar el resultado de salida `./tp0 <fileTest/testHigh.txt >fileTest/out.txt`

- *out.txt*:
  ```
  4 224.360000 51.130000 195.430000 182.690000 171.440000 66.880000
  283.600000 194.280000 196.020000 46.310000 346.250000 269.890000
  104.020000 79.390000 156.520000 155.380000
  3 340.520000 -93.020000 299.560000 355.140000 -320.740000 94.100000
  198.740000 -131.340000 135.480000
  ```

5. Conclusiones

En primera instancia el trabajo práctico nos permitió familiarizarnos con la sintaxis del lenguaje C y sus particularidades, asi como también con el manejo de una máquina virtual como GXemul y herramientas como LaTeX. El hecho de tener que crear conexiones como tuneles ssh para compilar y copiar el código fuente sobre el emulador nos permitió enriquecer nuestros conocimientos de Linux y BASH.

Compilar el código fuente sobre el emulador nos permitió observar como instrucciones escritas en un lenguaje de más alto nivel como C se traducen a instrucciones en MIPS32. Se puede apreciar la gran cantidad de instrucciones MIPS32 que se ejecutan por cada linea de código escrita en C.

Finalmente podemos decir que el desarrollo del trabajo nos sirvió como una primera aproximación a las herramientas que vamos a usar durante la materia, lo cual nos permitirá abordar los sucesivos trabajos prácticos con mayores conocimientos sin tener que preocuparnos por como utilizarlas sino más bien en como aplicar los conceptos.

6. Codigo C

6.1. Makefile

```
CFLAGS= -Wall -g  -std=c99
CC=gcc

all: tp0

tp0: tp0.c tp0.h
    $(CC) $(CFLAGS) tp0.c -o tp0
clean:
    rm tp0
```

6.2. tp0.h

```
#ifndef TP0_H_
#define TP0_H_
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>


typedef struct matrix {
    size_t rows;
    size_t cols;
    double* array;
} matrix_t;

// Constructor de matrix_t
matrix_t* create_matrix(size_t rows, size_t cols);

// Destructor de matrix_t
void destroy_matrix(matrix_t* m);

// Imprime matrix_t sobre el file pointer fp en el formato
    solicitado
// por el enunciado
int print_matrix(FILE* fd, matrix_t* m);

// Multiplica las matrices en m1 y m2
matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);

void printHelp();

int  checkArguments(int,char*[]);

/* Devuelve el primer elemento del archivo a leer, es
    decir la dimension*/
int leerDimensionMatriz();
```

```
    /* Lee matriz de una linea del stdin*/
    matrix_t* readMatrix(matrix_t* matrix);

    #endif /* TP0_H_ */
```

6.3. tp0.c

```
    #include "tp0.h"

void printHelp() {
    char *help = "Usage:"
            "\t ./tp0 -h \n"
            "\t ./tp0 -V \n"
            "\t ./tp0 < in_file > out_file \n"
            "Options: \n"
            " -V, --version \n\t Print version and quit. \n"
            " -h, --help \n\t Print this information and quit
                .\n"
            " Examples: \n"
            "  ./tp0 < in.txt > out.txt \n"
            "  cat in.txt | ./tp0 > out.txt \n";

    printf("%s", help);
}


// Devuelve mensaje de ayuda si el segundo argumento es -h o
    --help y
// la version si es -V, -v o --version. En cualquier otro caso
     devuelve 1.
int checkArguments(int cantidadArgumentos, char* argumentos[])
    {
    int retorno = 1;
    if ((cantidadArgumentos == 2)
            && ((strcmp(argumentos[1], "-h") == 0)
                    || (strcmp(argumentos[1], "--help") == 0))
                        ) {
        printHelp();
        retorno = 0;
    } else if ((cantidadArgumentos == 2)
            && (((strcmp(argumentos[1], "-V") == 0))
                    || (strcmp(argumentos[1], "--version") ==
                        0) || (strcmp(argumentos[1], "-v") ==
                        0))) {
        printf("Version 1.0\n");
        retorno = 0;
    }

    return retorno;
}


// Lee el primer elemento del stdin que corresponde a la
    dimension de la matriz cuadrada
```

```
int leerDimensionMatriz(){

    int dimension = 0;
    if(( fscanf(stdin, "%d", &dimension)==1) && !feof(stdin)){
        if (ferror (stdin)){
            printf ("Error reading stdin\n");
            exit(EXIT_FAILURE);
        }
    }
    return dimension;
}


// Constructor de matrix_t
matrix_t* create_matrix(size_t filas, size_t columnas){

    matrix_t* matriz;
    matriz = malloc(sizeof(matrix_t));
    (*matriz).array = malloc(sizeof(double)*filas*columnas);
    (*matriz).rows = filas;
    (*matriz).cols = columnas;

    return matriz;
}
// Carga la matrix previamente creada con floats de la matrix
matrix_t* readMatrix(matrix_t* matrix){
    double value = 0.0;
    int cantNums, j;

    cantNums = (*matrix).rows * (*matrix).rows;
    //m = (double*) malloc(sizeof(double)*cantNums);
    if(matrix == NULL){
        fprintf(stderr, "Error malloc \n");
    }

    for(j =0;j<cantNums; j++){
        if(fscanf(stdin, "%lf", &value) ==1){
            if (ferror (stdin)){
                printf ("Error reading stdin\n");
                if(matrix!=NULL){
                    free((*matrix).array);
                    free(matrix);
                    matrix = NULL;
                }
                exit(EXIT_FAILURE);
            }
            (*matrix).array[j] = value;
        }else{
            fprintf(stderr, "Matriz invÃ¡lida \n" );
            if(matrix!=NULL){
                free((*matrix).array);
                free(matrix);
                matrix = NULL;
            }
```

```
            exit(EXIT_FAILURE);
        }

    }
    return matrix;
}

// Multiplica las matrices en m1 y m2
matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2){

    int dimension = (*m1).rows;
    matrix_t* matrizResultado = create_matrix(dimension,
        dimension);
    int pos = 0;
    int i,j,k, m;

    for( i =0 ; i < dimension*dimension; i=i+dimension){
        for(k=0; k < dimension ; k++){
            double sum = 0.0;
            m = i;
            // printf("Entro for de K con k = %d\n",k );
            for(j=k; j < dimension*dimension; j=j+dimension){
                sum+= (*m1).array[m]*(*m2).array[j];
                m++;
            }
            (*matrizResultado).array[pos] = sum;
            pos++;
        }
    }
    return matrizResultado;
}

// Destructor de matrix_t
void destroy_matrix(matrix_t* matriz){
    if(matriz != NULL){
        free((*matriz).array);
        free(matriz);
        matriz = NULL;
    }
}


// Imprime matrix_t
int print_matrix(FILE* fd, matrix_t* matrix){
    int dim = (*matrix).rows;
    fprintf(fd, "%d ", dim);
    if (ferror(stdout)){
        fprintf(stderr, "Error printing stdout\n");
        destroy_matrix(matrix);
        exit(EXIT_FAILURE);
    }
    for(int i=0;i<dim*dim;i++){
        fprintf(fd, " %f",(*matrix).array[i]);
        if (ferror(stdout)){
```

```
            fprintf(stderr, "Error printing stdout\n");
            destroy_matrix(matrix);
            exit(EXIT_FAILURE);
        }
    }
    fprintf(fd, "\n");
    if (ferror(stdout)){
        fprintf(stderr, "Error printing stdout\n");
        destroy_matrix(matrix);
        exit(EXIT_FAILURE);
    }
    return 0;
}


// argc == argument count, argv== argument vector
 int main(int argc, char *argv[]) {

    if (!checkArguments(argc,argv)){
        return 1; // Error en los argumentos
    }else{
        do{
            int dimension = leerDimensionMatriz();
            if(feof(stdin)){
                return 0;
            }
            if(dimension > 0){
                matrix_t* matrix1 = create_matrix(dimension,
                    dimension);
                matrix1 = readMatrix(matrix1);
                matrix_t* matrix2 = create_matrix(dimension,
                    dimension);
                matrix2 = readMatrix(matrix2);
                matrix_t* matrizResultado = matrix_multiply(
                    matrix1, matrix2);

                 print_matrix(stdout, matrizResultado);

                //Borra las matrices creadas
                destroy_matrix(matrix1);
                destroy_matrix(matrix2);
                destroy_matrix(matrizResultado);
            }else{
                fprintf(stderr, "Error: Matriz invÃ¡lida para
                    multiplicaci\'on \n");
                exit(EXIT_FAILURE);
            }

        }while(!feof(stdin));
    }
    return 0;
}
```

7. Código MIPS32

7.1. tp0.S

Parte de código generado por MIPS32

```
    .file   1 "tp0.c"
    .section .mdebug.abi32
    .previous
    .abicalls
    .rdata
    .align  2

    ...

matrix_multiply:
    .frame  $fp,80,$ra      # vars= 40, regs= 3/0, args= 16,
       extra= 8
    .mask   0xd0000000,-8
    .fmask  0x00000000,0
    .set    noreorder
    .cpload $t9
    .set    reorder
    subu    $sp,$sp,80
    .cprestore 16
    sw  $ra,72($sp)
    sw  $fp,68($sp)
    sw  $gp,64($sp)
    move    $fp,$sp
    sw  $a0,80($fp)
    sw  $a1,84($fp)
    lw  $v0,80($fp)
    lw  $v0,0($v0)
    sw  $v0,24($fp)
    lw  $a0,24($fp)
    lw  $a1,24($fp)
    la  $t9,create_matrix
    jal $ra,$t9
    sw  $v0,28($fp)
    sw  $zero,32($fp)
    sw  $zero,36($fp)
$L40:
    lw  $v1,24($fp)
    lw  $v0,24($fp)
    mult    $v1,$v0
    mflo    $v1
    lw  $v0,36($fp)
    slt $v0,$v0,$v1
    bne $v0,$zero,$L43
    b   $L41
$L43:
    sw  $zero,44($fp)
$L44:
    lw  $v0,44($fp)
    lw  $v1,24($fp)
```

```
    slt  $v0,$v0,$v1
    bne  $v0,$zero,$L47
    b    $L42
$L47:
    sw   $zero,56($fp)
    sw   $zero,60($fp)
    lw   $v0,36($fp)
    sw   $v0,48($fp)
    lw   $v0,44($fp)
    sw   $v0,40($fp)
$L48:
    lw   $v1,24($fp)
    lw   $v0,24($fp)
    mult     $v1,$v0
    mflo     $v1
    lw   $v0,40($fp)
    slt  $v0,$v0,$v1
    bne  $v0,$zero,$L51
    b    $L49
$L51:
    lw   $a0,80($fp)
    lw   $v0,48($fp)
    sll  $v1,$v0,3
    lw   $v0,8($a0)
    addu     $a1,$v1,$v0
    lw   $a0,84($fp)
    lw   $v0,40($fp)
    sll  $v1,$v0,3
    lw   $v0,8($a0)
    addu     $v0,$v1,$v0
    l.d  $f2,0($a1)
    l.d  $f0,0($v0)
    mul.d    $f2,$f2,$f0
    l.d  $f0,56($fp)
    add.d    $f0,$f0,$f2
    s.d  $f0,56($fp)
    lw   $v0,48($fp)
    addu     $v0,$v0,1
    sw   $v0,48($fp)
    lw   $v1,40($fp)
    lw   $v0,24($fp)
    addu     $v0,$v1,$v0
    sw   $v0,40($fp)
    b    $L48
$L49:
    lw   $a0,28($fp)
    lw   $v0,32($fp)
    sll  $v1,$v0,3
    lw   $v0,8($a0)
    addu     $v0,$v1,$v0
    l.d  $f0,56($fp)
    s.d  $f0,0($v0)
    lw   $v0,32($fp)
    addu     $v0,$v0,1
```

```
    sw   $v0,32($fp)
    lw   $v0,44($fp)
    addu     $v0,$v0,1
    sw   $v0,44($fp)
    b    $L44
$L42:
    lw   $v0,36($fp)
    lw   $v1,24($fp)
    addu     $v0,$v0,$v1
    sw   $v0,36($fp)
    b    $L40
$L41:
    lw   $v0,28($fp)
    move     $sp,$fp
    lw   $ra,72($sp)
    lw   $fp,68($sp)
    addu     $sp,$sp,80
    j    $ra
    .end     matrix_multiply
```

Referencias

[1] .