



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires

Organización de Computadoras 66.20

Trabajo practico N°1

Integrantes:

- Alvarez Fernando 99373
- Braccelarghe Ailin 99366
- Suarez Martin 101540

1 Documentación del diseño e implementación del programa

Este trabajo practico se basa en codear la función *matrix_multiply* en código MIPS32 y que funcione con el resto del programa en C.

Para hacer el traspaso de lenguaje, tuvimos que en la función *matrix_multiply* hacer las funciones *index_value* y *value_obtain* son implementadas internamente de la función, para no tener que hacer funciones extras en MIPS32.

Para realizar la función, se hizo uso de las funciones *mymalloc* y *myfree* proporcionadas por la cátedra, las cuales reemplazaran al *malloc* y al *free* de C.

1.1 Estructura del programa

- main.c
- matrix.h
- matrix.c
- matrix_multiply.S
- matrix_multiply.h
- matrix_destroy.S
- matrix_destroy.h
- mymalloc.S
- mymalloc.h
- makefile

1.2 Estructura de matrix_t

- `matrix_t* create_matrix(size_t rows, size_t cols);`

Es el constructor de la matriz.

- `int index_value(matrix_t* m, int x, int y, double value);`

Sirve para insertar un valor (*double*) en la coordenada (x,y) de la matriz, donde **x** indica la columna e **y** la fila

- `double value_obtain(matrix_t* m, int x, int y);`

Permite obtener el valor que se encuentra en la coordenada (x,y) de la matriz

- `int complete_matrix(double* values, matrix_t* m);`

Completa la matriz a partir de un array de *double*'s

- `int destroy_matrix(matrix_t* m);`

Destructor de la matriz

- `int print_matrix(FILE* fp, matrix_t* m);`

Se encarga de imprimir la matriz resultante en un file pointer dado

- `//matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);`

Se encarga de multiplicar dos matrices, devuelve una matriz resultado

2 Comandos del makefile

\$ make build: Compila el programa.

\$ make run: Ejecuta el programa.

\$ make all: Compila y ejecuta el programa.

\$ make assembly: Compila el programa sin optimizaciones y se detiene al generar el código assembly. Utiliza además el parámetro -mrnames para que el compilador reemplace los números de registro por sus nombres.

3 Corrida de pruebas

3.1 Pruebas por entrada de consola

- **Propósito:** Funcionamiento correcto de la multiplicación para N=2
Resultado esperado: 2 7 10 15 22
Resultado obtenido: 2 7 10 15 22
main in free(): error: junk pointer, too high to make sense
main in free(): error: junk pointer, too high to make sense
Acción: Ante estos mensajes, vimos que se debían a que como la matriz resultado era generada con *mymalloc*, el destructor de la misma tendría que ser implementado con *myfree* y no con el *free* de C, por lo tanto implementamos la función *matrix_destroy* en MIPS32.
- **Propósito:** Funcionamiento de la función *matrix_destroy*
Resultado esperado: 2 7 10 15 22
Resultado obtenido: 2 7 10 15 22
- **Propósito:** Reconocimiento de caracteres fuera del formato establecido
Resultado esperado: Cierre del programa
Resultado obtenido: 'ERROR EN LECTURA: Valor en formato erróneo'
- **Propósito:** Reconocimiento de caracteres fuera del formato establecido
Resultado esperado: Cierre del programa
Resultado obtenido: 'ERROR EN LECTURA: Valor en formato erróneo'
- **Propósito:** Detección de mal tamaño de la matriz
Resultado esperado: Cierre del programa
Resultado obtenido: 'ERROR EN LECTURA: No se recibió un tamaño correcto de matriz'
- **Propósito:** Detección de mal tamaño de la matriz
Resultado esperado: Cierre del programa
Resultado obtenido: 'ERROR EN LECTURA: No se recibió un tamaño correcto de matriz'
- **Propósito:** Detección de mal tamaño de la matriz
Resultado esperado: Cierre del programa
Resultado obtenido: 'ERROR EN LECTURA: No se recibió un tamaño correcto de matriz'
- **Propósito:** Detección correcta del formato exponencial
Resultado esperado: 2 9e+07 26200 1.8e+10 2.524e+07
Resultado obtenido: 2 9e+07 26200 1.8e+10 2.524e+07

- **Propósito:** Funcionamiento correcto de la multiplicación para $N=3$
Resultado esperado: 3 30 36 42 66 81 96 102 126 150
Resultado obtenido: 3 30 36 42 66 81 96 102 126 150

Y su salida por un txt, luego de realizar cada uno de estos test, seria:

```
out.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
Resultado de la multiplicacion:
| 7 10 | 15 22
Resultado de la multiplicacion:
| 7 10 | 15 22
Resultado de la multiplicacion:
| 9e+07 26200 | 1.8e+10 2.524e+07
Resultado de la multiplicacion:
| 30 36 42 | 66 81 96 | 102 126 150
```

3.2 Entrada por txt

El txt utilizado fue: **test.txt** con lo siguiente:

```
test.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
2 1 2 3 4 1 2 3 4
2 1 1 1 1 1 1 1
3 1 2 3 4 5 6.1 3 2 1 1 0 0 0 1 0 0 0 1
5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1
3 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
```

Su salida en forma de archivo fue la siguiente:

```
out.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
Resultado de la multiplicacion:
| 7 10 | 15 22
Resultado de la multiplicacion:
| 2 2 | 2 2
Resultado de la multiplicacion:
| 1 2 3 | 4 5 6.1 | 3 2 1
Resultado de la multiplicacion:
| 215 230 245 260 275 | 490 530 570 610 650 | 765 830 895 960 1025 | 1040 1130 1220 1310 1400 | 1315 1430 1545 1660 1775
Resultado de la multiplicacion:
| 3.33761e-308 3.33761e-308 | 3.33761e-308 3.33761e-308
```

Como se puede ver la ultima multiplicación que pudo realizar es la multiplicación de matrices nulas, dado que la siguiente linea tiene un formato incorrecto, por lo cual el programa corta. En la salida se puede ver como el emulador nos da un numero demasiado chico, aproximado a 0.

Su salida por consola es:

```
1 2 7 10 15 22
2 2 2 2 2
3 1 2 3 4 5 6.1 3 2 1
4 5 215 230 245 260 275 490 530 570 610 650 765 830 895 960 1025 1040 1130 1220 1310 1400 1315
1430 1545 1660 1775
5 2 3.33761e-308 3.33761e-308 3.33761e-308 3.33761e-308
6 ERROR EN LECTURA: Faltan valores
```

4 Código fuente

4.1 main.c

```
1 #include "matrix.h"
2 #include "matrix_multiply.h"
3 #include "matrix_destroy.h"
4 #include <stdlib.h>
5 #include <stdio.h>
6 #include <string.h>
7
8 int main(int argc, const char* argv[])
9 {
10     int size;
11     double value;
12     char buffer[4096];
13     char* delim = " ";
14     int ok;
15     int error = 0;
16
17
18     while(fgets(buffer, sizeof(buffer), stdin)){
19
20         ok = sscanf(buffer, "%i", &size);
21         if(!ok || size <= 0){
22             fprintf(stderr, "ERROR EN LECTURA: No se recibio un tama o correcto de matriz\n");
23             exit(1);
24         }
25
26         int ammount = 2*size*size;
27         double* values = (double*) malloc(ammount*sizeof(double));
28         if(!values) {
29             fprintf(stderr, "MEMORY ERROR\n");
30             exit(1);
31         }
32         char* string = strtok(buffer, delim);
33         for(int i = 0; i < ammount; i++){
34             string = strtok(NULL, delim);
35             if(!string){
36                 free(values);
37                 fprintf(stderr, "%s", "ERROR EN LECTURA: Faltan valores\n");
38                 exit(1);
39             }
40
41             char* error = "";
42             value = strtod(string, &error);
43             if(strcmp("", error) != 0 && strcmp("\n", error) != 0){
44                 free(values);
45                 fprintf(stderr, "%s", "ERROR EN LECTURA: Valor en formato erroneo\n");
46                 exit(1);
47             }
48
49             values[i] = value;
50         }
51
52
53         double* values_A = (double*) malloc(size*size*sizeof(double));
54         if(!values_A){
55             fprintf(stderr, "%s", "ERROR AL CARGAR VALOR A" );
56             free(values);
57             exit(1);
58         }
59         memcpy(values_A, values, size*size*sizeof(double));
60
61         double* values_B = (double*) malloc(size*size*sizeof(double));
62         if(!values_B){
63             fprintf(stderr, "%s", "ERROR AL CARGAR VALOR B" );
64             free(values);
```

```

65     free(values_A);
66     exit(1);
67 }
68 memcpy(values_B, values+size*size, size*size*sizeof(double));
69
70 matrix_t* matrix_A = create_matrix(size, size);
71 if(!matrix_A){
72     fprintf(stderr, "%s", "ERROR AL CREAR MATRIZ A" );
73     free(values);
74     free(values_A);
75     free(values_B);
76     exit(1);
77 }
78 matrix_t* matrix_B = create_matrix(size, size);
79 if(!matrix_B){
80     fprintf(stderr, "%s", "ERROR AL CREAR MATRIZ B" );
81     free(values);
82     free(values_A);
83     free(values_B);
84     destroy_matrix(matrix_A);
85     exit(1);
86 }
87
88 int comp = complete_matrix(values_A, matrix_A);
89 if(comp == -1){
90     fprintf(stderr, "%s", "ERROR AL COMPLETAR MATRIZ A" );
91     free(values);
92     free(values_A);
93     free(values_B);
94     destroy_matrix(matrix_A);
95     destroy_matrix(matrix_B);
96     exit(1);
97 }
98
99 comp = complete_matrix(values_B, matrix_B);
100 if(comp == -1){
101     fprintf(stderr, "%s", "ERROR AL COMPLETAR MATRIZ B" );
102     free(values);
103     free(values_A);
104     free(values_B);
105     destroy_matrix(matrix_A);
106     destroy_matrix(matrix_B);
107     exit(1);
108 }
109
110 matrix_t* matrix_C = matrix_multiply(matrix_A, matrix_B);
111 if(!matrix_C) {
112     fprintf(stderr, "%s", "ERROR AL REALIZAR MULTIPLICACION" );
113     free(values);
114     free(values_A);
115     free(values_B);
116     destroy_matrix(matrix_A);
117     destroy_matrix(matrix_B);
118     exit(1);
119 }
120
121 FILE *file;
122 file = fopen("out.txt", "a");
123 if(!file){
124     fprintf(stderr, "%s", "ERROR AL LEER ARCHIVO SALIDA" );
125     error = 1;
126 }
127
128 else{
129     comp = print_matrix(file, matrix_C);
130     if(comp == -1) error = 1;
131 }
132
133 free(values);
134 free(values_A);

```

```

135     free(values_B);
136
137     destroy_matrix(matrix_A);
138     destroy_matrix(matrix_B);
139     matrix_destroy(matrix_C);
140
141     fclose(file);
142 }
143 exit(error);
144 }

```

4.2 matrix.c

```

1  #include "matrix.h"
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5
6  struct matrix{
7      size_t rows;
8      size_t cols;
9      double* array;
10 };
11
12 matrix_t* create_matrix(size_t rows, size_t cols){
13     if(rows != cols) return NULL;
14     matrix_t* matrix = malloc(sizeof(matrix_t));
15     if(!matrix) {
16         fprintf(stderr, "MEMORY ERROR");
17         return NULL;
18     }
19     matrix->rows = rows;
20     matrix->cols = cols;
21     matrix->array = (double*)calloc(cols*rows, sizeof(double));
22     if(!matrix->array) {
23         free(matrix);
24         fprintf(stderr, "MEMORY ERROR");
25         return NULL;
26     }
27     return matrix;
28 }
29
30 int index_value(matrix_t* m, int x, int y, double value){
31     if(m->rows == 0 || m->cols == 0) return -1;
32     m->array[y*m->rows + x] = value;
33     return 0;
34 }
35
36 double value_obtain(matrix_t* m, int x, int y){
37     return m->array[y*m->rows + x];
38 }
39
40 int complete_matrix(double* values, matrix_t* m){
41     if(!m) {
42         fprintf(stderr, "NO MATRIX CREATED ERROR");
43         return -1;
44     }
45     for(int y = 0; y < m->rows; y++){
46         for(int x = 0; x < m->cols; x++){
47             double value = values[m->rows * y + x];
48             int comp = index_value(m, x, y, value);
49             if(comp == -1) return -1;
50         }
51     }
52     return 0;
53 }
54
55 int destroy_matrix(matrix_t* m){

```

```

56     if (!m) {
57         fprintf(stderr, "NO MATRIX CREATED ERROR");
58         return -1;
59     }
60     free(m->array);
61     free(m);
62     return 0;
63 }
64
65 int print_matrix(FILE* fp, matrix_t* m){
66     if (!m) {
67         fprintf(stderr, "NO MATRIX CREATED ERROR");
68         return (-1);
69     }
70     if (!fp){
71         perror("Error al crear el archivo de salida");
72         return (-1);
73     }
74     else{
75         fprintf(stdout, "%d ", m->rows );
76         fprintf(fp, "%s\n", "Resultado de la multiplicacion:" );
77         for(int y = 0; y < m->rows; y++){
78             fputc('|', fp);
79             fputc(' ', fp);
80             for(int x = 0; x < m->cols; x++){
81                 double value = value_obtain(m, x, y);
82                 fprintf(stdout, "%lg ", value);
83                 fprintf(fp, "%lg", value );
84                 fputc(' ', fp);
85             }
86         }
87         fprintf(stdout, "\n");
88         fputc('\n', fp);
89     }
90     fflush(fp);
91     return 0;
92 }
93
94 /*matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2){
95     if(m1->rows != m1->cols || m2->rows != m2->cols || m1->rows != m2->rows) {
96         fprintf(stderr, "DIMENSION ERROR");
97         return NULL;
98     }
99     matrix_t* mresult = create_matrix(m1->rows, m1->cols);
100     if(!mresult) {
101         fprintf(stderr, "NO MATRIX CREATED ERROR");
102         return NULL;
103     }
104     int N = m1->cols;
105     for(int i = 0; i < N; i++){
106         for(int x = 0; x < N; x++){
107             index_value(mresult, x, i, 0);
108             for(int y = 0; y < N; y++){
109                 double value = value_obtain(mresult, x, i);
110                 value += value_obtain(m1, y, i) * value_obtain(m2, x, y);
111                 index_value(mresult, x, i, value);
112             }
113         }
114     }
115     return mresult;
116 }*/

```


4.3 matrix.h

```
1
2 #ifndef MATRIX_H
3 #define MATRIX_H
4
5 #include <stdbool.h>
6 #include <stdlib.h>
7 #include <stdio.h>
8
9 struct matrix;
10 typedef struct matrix matrix_t;
11
12 // Constructor de matriz.
13 matrix_t* create_matrix(size_t rows, size_t cols);
14
15 // Guarda un double en la coordenada (x,y) dada.
16 int index_value(matrix_t* m, int x, int y, double value);
17
18 // Obtiene el valor (double) en la coordenada (x,y).
19 double value_obtain(matrix_t* m, int x, int y);
20
21 // Completa la matriz a partir de un array de doubles.
22 int complete_matrix(double* values, matrix_t* m);
23
24 // Destructor de matriz
25 int destroy_matrix(matrix_t* m);
26
27 // Imprime matriz sobre el file pointer fp.
28 int print_matrix(FILE* fp, matrix_t* m);
29
30 // Multiplica las matrices en m1 y m2.
31 //matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);
32
33 #endif // MATRIX_H
```

4.4 matrix_destroy.h

```
1 #ifndef _MATRIX_DESTROY_H_INCLUDED_
2 #define _MATRIX_DESTROY_H_INCLUDED_
3
4 extern int matrix_destroy(matrix_t* m);
5
6 #endif
```

4.5 matrix_multiply.h

```
1 #ifndef _MATRIX_MULTIPLY_H_INCLUDED_
2 #define _MATRIX_MULTIPLY_H_INCLUDED_
3
4 extern matrix_t *matrix_multiply(matrix_t* m1, matrix_t* m2);
5
6 #endif
```

4.6 mymalloc.h

```
1 #ifndef _MYMALLOC_H_INCLUDED_
2 #define _MYMALLOC_H_INCLUDED_
3
4 extern void *mymalloc(size_t);
5 extern void myfree(void *);
6
7 #endif
```

5 Código Assembly MIPS32

5.1 matrix_multiply.S

```
1 #include <mips/regdef.h>
2 #include <sys/syscall.h>
3
4 .text
5 .abicalls
6 .align 2
7 .globl matrix_multiply
8 .ent matrix_multiply
9
10 matrix_multiply:
11
12 .frame $fp, 40, ra
13 .set noreorder # apaga reordenamiento de instrucciones
14 .cpload t9 # directiva usada para codigo PIC
15 .set reorder
16 subu sp, sp, 40 # 4 (SRA) + 2 (LTA) + 4 (ABA)
17 .cpstore 24 # inserta aqui "sw gp, 24(sp)", mas "lw gp, 24(sp)" luego de cada jal
    salvado de callee-saved regs en SRA
18 sw $fp, 28(sp)
19 sw ra, 32(sp)
20
21 move $fp, sp # de aqui al fin de la funcion uso $fp en lugar de sp.
22 sw a0, 40($fp) # guardo en el stack a0 = *m1
23 sw a1, 44($fp) # guardo en el stack a1 = *m2
24
25 lw t0, 0(a0) #t0 = m1->rows
26 lw t1, 4(a0) #t1 = m1->cols
27 bne t0, t1, _dimension_error # m1->rows != m1->cols
28
29 lw t2, 0(a1) #t2 = m2->rows
30 lw t3, 4(a1) #t3 = m2->cols
31 bne t2, t3, _dimension_error #m2->rows != m2->cols
32 bne t0, t3, _dimension_error #m1->rows != m2->rows
33
34 li a0, 12
35 la t9, mymalloc
36 jal ra, t9
37 move s0, v0 #guardo el puntero de mresult en s0
38 beq s0, -1, _malloc_matrix_error
39
40 lw a0, 40($fp) # a0 = *matriz1
41 lw a1, 44($fp) #a1 = *matriz2
42 lw t0, 0(a0) #t0 = m1->rows
43 lw t1, 4(a0) #t1 = m1->cols
44
45 sw t0, 0(s0) #le asigno las rows a la nueva matriz
46 sw t1, 4(s0) #le asigno las cols a la nueva matriz creo array de la nueva matriz
47 mul t0, t0, t1
48 sll t0, t0, 3
49 move a0, t0 #guardo en a0 el valor de t0(guardo el size que quiero del array)
50 la t9, mymalloc
51 jal ra, t9
52 move s1, v0
53 beq s1, -1, _malloc_array_error
54 sw s1, 8(s0)
55 sw s0, 16($fp)
56
57 lw a0, 40($fp) #cargo a0 (m1)
58 lw a1, 44($fp) #cargo a1 (m2)
59 lw t0, 0(a0) #t1 = m1->rows = N
60
61 move t1, zero #i=0
62 for_i:
63 move t2, zero # x=0: t3
```

```

64 for_x:
65     lw     t4, 8(s0) #t4 = mresult->array
66     lw     t5, 0(s0) #t5 = mresult->rows
67     mul    t5, t5, t1 #t5 = mresult->rows * i
68     addu   t5, t5, t2 #t5 = mresult->rows * i + x
69
70     sll    t5, t5, 3 #multiplico por 8 porque son doubles
71     addu   t4, t4, t5
72     sw     zero, 0(t4)
73
74     move   t3, zero # y=0: t4
75 for_y:
76
77     lw     t4, 8(a0) #t4 = m1->array
78     lw     t5, 0(a0) #t5 = m1->rows
79     mul    t5, t5, t1 #t5 = m1->rows * i
80     addu   t5, t5, t3 #t5 = m1->rows * i + y
81
82     sll    t5, t5, 3 #multiplico por 8 porque son doubles
83     addu   t4, t4, t5
84     l.d    $f0, 0(t4) #creo que en f0 se guarda m1->array[i*m1->rows + y]
85
86     lw     t4, 8(a1) #t4 = m2->array
87     lw     t5, 0(a1) #t5 = m2->rows
88     mul    t5, t5, t3 #t5 = m2->rows * y
89     addu   t5, t5, t2 #t5 = m2->rows * y + x
90
91     sll    t5, t5, 3 #multiplico por 8 porque son doubles
92     addu   t4, t4, t5
93     l.d    $f2, 0(t4) #creo que en f1 se guarda m2->array[y*m2->rows + x]
94
95     mul.d   $f0, $f2, $f0 # multiplicacion de valores de las matrices
96
97     lw     t4, 8(s0) #t4 = mresult->array
98     lw     t5, 0(s0) #t5 = mresult->rows
99     mul    t5, t5, t1 #t5 = mresult->rows * i
100    addu   t5, t5, t2 #t5 = mresult->rows * i + x
101
102    sll    t5, t5, 3 #multiplico por 8 porque son doubles
103    addu   t4, t4, t5
104    l.d    $f2, 0(t4)
105    add.d   $f0, $f0, $f2
106    s.d     $f0, 0(t4) # mresult->array[t7] = multiplicacion
107
108    addiu   t3, t3, 1 # y++
109    sltu   t6, t3, t0 # si y < N t6 = 1 si no t6 = 0
110    bnez   t6, for_y #salta al for si t6 != 0
111
112    addiu   t2, t2, 1 # x++
113    sltu   t6, t2, t0
114    bnez   t6, for_x
115
116    addiu   t1, t1, 1 #i++
117    sltu   t6, t1, t0
118    bnez   t6, for_i
119
120    lw     v0, 16($fp) #con esto hacemos el return del puntero de mresult
121    lw     ra, 32(sp)
122    lw     $fp, 28(sp)
123    lw     gp, 24(sp)
124    jr     ra
125
126 _created_matrix_error:
127     li     a0, 2 #File descriptor del write, modo stderr
128     la     a1, CREATEDMATRIX_ERROR
129     li     a2, 23
130     li     v0, SYS_write
131     syscall
132     li     v0, 0 #return NULL
133     lw     ra, 32(sp)

```

```

134 lw    $fp, 28(sp)
135 lw    gp, 24(sp)
136 jr    ra
137
138
139 _dimension_error:
140     li    a0, 2 #File descriptor del write, modo stderr
141     la    a1, DIMERROR
142     li    a2, 15
143     li    v0, SYS_write
144     syscall
145     li    v0, 0 #return NULL
146     lw    ra, 32(sp)
147     lw    $fp, 28(sp)
148     lw    gp, 24(sp)
149     jr    ra
150
151
152 _malloc_matrix_error:
153     li    a0, 2 #File descriptor del write, modo stderr
154     la    a1, MALLOC_MATRIX_ERROR
155     li    a2, 49
156     li    v0, SYS_write
157     syscall
158     li    v0, 0 #return NULL
159     lw    ra, 32(sp)
160     lw    $fp, 28(sp)
161     lw    gp, 24(sp)
162     jr    ra
163
164 _malloc_array_error:
165     move  a0, s0
166     la    t9, myfree
167     jal   ra, t9
168
169     li    a0, 2 #File descriptor del write, modo stderr
170     la    a1, MALLOC_ARRAY_ERROR
171     li    a2, 48
172     li    v0, SYS_write
173     syscall
174     li    v0, 0 #return NULL
175     lw    ra, 32(sp)
176     lw    $fp, 28(sp)
177     lw    gp, 24(sp)
178     jr    ra
179
180 .end matrix_multiply
181
182 .size matrix_multiply,.-matrix_multiply
183
184 .rdata
185 .align 2
186
187 DIMERROR: .ascii "DIMENSION ERROR"
188 CREATED_MATRIX_ERROR: .ascii "NO MATRIX CREATED ERROR"
189 MALLOC_MATRIX_ERROR: .ascii "ERROR AL GUARDAR MEMORIA PARA LA MATRIZ RESULTADO"
190 MALLOC_ARRAY_ERROR: .ascii "ERROR AL GUARDAR MEMORIA PARA EL ARRAY RESULTADO"

```

5.2 matrix_destroy.S

```
1
2 #include <sys/syscall.h>
3 #include <mips/regdef.h>
4
5 .text
6 .abicalls
7 .align 2
8 .globl matrix_destroy
9 .ent matrix_destroy
10
11 matrix_destroy:
12
13 .frame $fp, 40, ra
14 .set noreorder # apaga reordenamiento de instrucciones
15 .cpload t9 # directiva usada para codigo PIC
16 .set reorder
17 subu sp, sp, 40 # 4 (SRA) + 2 (LTA) + 4 (ABA)
18 .cprestore 24 # inserta aqui "sw gp, 24(sp)", mas "lw gp, 24(sp)" luego de cada jal
    salvado de callee-saved regs en SRA
19 sw $fp, 28(sp)
20 sw ra, 32(sp)
21
22 move $fp, sp # de aqui al fin de la funcion uso $fp en lugar de sp.
23 sw a0, 40($fp) # guardo en el stack a0 = *m
24
25 beqz a0, _matrix_created_error
26 ld a0, 8(a0)
27 la t9, myfree
28 jal ra, t9
29 ld a0, 40($fp)
30 la t9, myfree
31 jal ra, t9
32 li v0, 0 #return 0
33 lw ra, 32(sp)
34 lw $fp, 28(sp)
35 lw gp, 24(sp)
36 jr ra
37
38 _matrix_created_error:
39 li a0, 2 #File descriptor del write, modo stderr
40 la a1, CREATED_MATRIX_ERROR
41 li a2, 23
42 li v0, SYS_write
43 syscall
44 li v0, -1 #return -1
45 lw ra, 32(sp)
46 lw $fp, 28(sp)
47 lw gp, 24(sp)
48 jr ra
49
50 .end matrix_destroy
51
52 .size matrix_destroy,.-matrix_destroy
53
54 .rdata
55 .align 2
56
57 CREATED_MATRIX_ERROR: .asciiz "NO MATRIX CREATED ERROR"
```

5.3 mymalloc.S

```
1
2 #include <sys/syscall.h>
3 #include <mips/regdef.h>
4
5 #define MYMALLOC_SIGNATURE 0xdeadbeef
6
7 #ifndef PROT_READ
8 #define PROT_READ 0x01
9 #endif
10
11 #ifndef PROT_WRITE
12 #define PROT_WRITE 0x02
13 #endif
14
15 #ifndef MAP_PRIVATE
16 #define MAP_PRIVATE 0x02
17 #endif
18
19 #ifndef MAP_ANON
20 #define MAP_ANON 0x1000
21 #endif
22
23 .text
24 .align 2
25 .globl mymalloc
26 .ent mymalloc
27 mymalloc:
28     subu    sp, sp, 56
29     sw      ra, 48(sp)
30     sw      $fp, 44(sp)
31     sw      a0, 40(sp) # Temporary: original allocation size.
32     sw      a0, 36(sp) # Temporary: actual allocation size.
33     li      t0, -1
34     sw      t0, 32(sp) # Temporary: return value (defaults to -1).
35 #if 0
36     sw      a0, 28(sp) # Argument building area (#8?).
37     sw      a0, 24(sp) # Argument building area (#7?).
38     sw      a0, 20(sp) # Argument building area (#6).
39     sw      a0, 16(sp) # Argument building area (#5).
40     sw      a0, 12(sp) # Argument building area (#4, a3).
41     sw      a0, 8(sp)  # Argument building area (#3, a2).
42     sw      a0, 4(sp)  # Argument building area (#2, a1).
43     sw      a0, 0(sp)  # Argument building area (#1, a0).
44 #endif
45     move    $fp, sp
46
47     ## Adjust the original allocation size to a 4-byte boundary.
48
49     lw      t0, 40(sp)
50     addiu   t0, t0, 3
51     and     t0, t0, 0xffffffffc
52     sw      t0, 40(sp)
53
54     ## Increment the allocation size by 12 units, in order to
55     ## make room for the allocation signature, block size and
56     ## trailer information.
57
58     lw      t0, 40(sp)
59     addiu   t0, t0, 12
60     sw      t0, 36(sp)
61
62     ## mmap(0, sz, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANON, -1, 0)
63
64     li      v0, SYS_mmap
65     li      a0, 0
66     lw      a1, 36(sp)
67     li      a2, PROT_READ|PROT_WRITE
```

```

68  li  a3, MAP_PRIVATE|MAP_ANON
69
70  ## According to mmap(2), the file descriptor
71  ## must be specified as -1 when using MAP_ANON.
72
73  li  t0, -1
74  sw  t0, 16(sp)
75
76  ## Use a trivial offset.
77
78  li  t0, 0
79  sw  t0, 20(sp)
80
81  ## XXX TODO.
82
83  sw  zero, 24(sp)
84  sw  zero, 28(sp)
85
86  ## Execute the syscall, save the return value.
87
88  syscall
89  sw  v0, 32(sp)
90  beqz v0, mymalloc_return
91
92  ## Success. Check out the allocated pointer.
93
94  lw  t0, 32(sp)
95  li  t1, MYMALLOC_SIGNATURE
96  sw  t1, 0(t0)
97
98  ## The actual allocation size goes right after the signature.
99
100 lw  t0, 32(sp)
101 lw  t1, 36(sp)
102 sw  t1, 4(t0)
103
104 ## Trailer information.
105
106 lw  t0, 36(sp) # t0: actual allocation size.
107 lw  t1, 32(sp) # t1: Pointer.
108 addu t1, t1, t0 # t1 now points to the trailing 4-byte area.
109 xor t2, t0, MYMALLOC_SIGNATURE
110 sw  t2, -4(t1)
111
112 ## Increment the result pointer.
113
114 lw  t0, 32(sp)
115 addiu t0, t0, 8
116 sw  t0, 32(sp)
117
118 mymalloc_return:
119 ## Restore the return value.
120
121 lw  v0, 32(sp)
122
123 ## Destroy the stack frame.
124
125 move sp, $fp
126 lw  ra, 48(sp)
127 lw  $fp, 44(sp)
128 addu sp, sp, 56
129
130 j  ra
131 .end  mymalloc
132
133 .globl myfree
134 .ent  myfree
135 myfree:
136 subu sp, sp, 40
137 sw  ra, 32(sp)

```

```

138 sw $fp, 28(sp)
139 sw a0, 24(sp) # Temporary: argument pointer.
140 sw a0, 20(sp) # Temporary: actual mmap(2) pointer.
141 move $fp, sp
142
143 ## Calculate the actual mmap(2) pointer.
144
145 lw t0, 24(sp)
146 subu t0, t0, 8
147 sw t0, 20(sp)
148
149 ## XXX Sanity check: the argument pointer must be checked
150 ## in before we try to release the memory block.
151
152 ## First, check the allocation signature.
153
154 lw t0, 20(sp) # t0: actual mmap(2) pointer.
155 lw t1, 0(t0)
156 bne t1, MYMALLOC.SIGNATURE, myfree_die
157
158 ## Second, check the memory block trailer.
159
160 lw t0, 20(sp) # t0: actual mmap(2) pointer.
161 lw t1, 4(t0) # t1: actual mmap(2) block size.
162 addu t2, t0, t1 # t2: trailer pointer.
163 lw t3, -4(t2)
164 xor t3, t3, t1
165 bne t3, MYMALLOC.SIGNATURE, myfree_die
166
167 ## All checks passed. Try to free this memory area.
168
169 li v0, SYS_munmap
170 lw a0, 20(sp) # a0: actual mmap(2) pointer.
171 lw a1, 4(a0) # a1: actual allocation size.
172 syscall
173
174 ## Bail out if we cannot unmap this memory block.
175
176 bnez v0, myfree_die
177
178 ## Success.
179
180 j myfree_return
181
182 myfree_die:
183 ## Generate a segmentation fault by writing to the first
184 ## byte of the address space (a.k.a. the NULL pointer).
185
186 sw t0, 0(zero)
187
188 myfree_return:
189 ## Destroy the stack frame.
190
191 move sp, $fp
192 lw ra, 32(sp)
193 lw $fp, 28(sp)
194 addu sp, sp, 40
195
196 j ra
197 .end myfree

```


6 Codigo MIPS32 generado por el compilador

6.1 main.s

```
1 .file 1 "main.c"
2 .section .mdebug.abi32
3 .previous
4 .abicalls
5 .rdata
6 .align 2
7 $LC0:
8 .ascii "\000"
9 .align 2
10 $LC1:
11 .ascii "%i\000"
12 .align 2
13 $LC2:
14 .ascii "ERROR EN LECTURA: No se recibio un tama\303\261o correct"
15 .ascii "o de matriz\n\000"
16 .align 2
17 $LC3:
18 .ascii "MEMORY ERROR\n\000"
19 .align 2
20 $LC4:
21 .ascii "%s\000"
22 .align 2
23 $LC5:
24 .ascii "ERROR EN LECTURA: Faltan valores\n\000"
25 .align 2
26 $LC6:
27 .ascii "\000"
28 .align 2
29 $LC7:
30 .ascii "\n\000"
31 .align 2
32 $LC8:
33 .ascii "ERROR EN LECTURA: Valor en formato erroneo\n\000"
34 .align 2
35 $LC9:
36 .ascii "ERROR AL CARGAR VALOR A\000"
37 .align 2
38 $LC10:
39 .ascii "ERROR AL CARGAR VALOR B\000"
40 .align 2
41 $LC11:
42 .ascii "ERROR AL CREAR MATRIZ A\000"
43 .align 2
44 $LC12:
45 .ascii "ERROR AL CREAR MATRIZ B\000"
46 .align 2
47 $LC13:
48 .ascii "ERROR AL COMPLETAR MATRIZ A\000"
49 .align 2
50 $LC14:
51 .ascii "ERROR AL COMPLETAR MATRIZ B\000"
52 .align 2
53 $LC15:
54 .ascii "ERROR AL REALIZAR MULTIPLICACION\000"
55 .align 2
56 $LC16:
57 .ascii "out.txt\000"
58 .align 2
59 $LC17:
60 .ascii "a\000"
61 .align 2
62 $LC18:
63 .ascii "ERROR AL LEER ARCHIVO SALIDA\000"
64 .text
```

6.2 matrix.s

```
1  .file 1 "matrix.c"
2  .section .mdebug.abi32
3  .previous
4  .abicalls
5  .rdata
6  .align 2
7  $LC0:
8  .ascii "MEMORY ERROR\000"
9  .text
10 .align 2
11 .globl create_matrix
12 .ent create_matrix
13 create_matrix:
14 .frame $fp,48,$ra # vars= 8, regs= 4/0, args= 16, extra= 8
15 .mask 0xd0010000,-4
16 .fmask 0x00000000,0
17 .set noreorder
18 .cpload $t9
19 .set reorder
20 subu $sp,$sp,48
21 .cpstore 16
22 sw $ra,44($sp)
23 sw $fp,40($sp)
24 sw $gp,36($sp)
25 sw $s0,32($sp)
26 move $fp,$sp
27 sw $a0,48($fp)
28 sw $a1,52($fp)
29 lw $v1,48($fp)
30 lw $v0,52($fp)
31 beq $v1,$v0,$L18
32 sw $zero,28($fp)
33 b $L17
34 $L18:
35 li $a0,12 # 0xc
36 la $t9, malloc
37 jal $ra,$t9
38 sw $v0,24($fp)
39 lw $v0,24($fp)
40 bne $v0,$zero,$L19
41 la $a0,--sF+176
42 la $a1,$LC0
43 la $t9,fprintf
44 jal $ra,$t9
45 sw $zero,28($fp)
46 b $L17
47 $L19:
48 lw $v1,24($fp)
49 lw $v0,48($fp)
50 sw $v0,0($v1)
51 lw $v1,24($fp)
52 lw $v0,52($fp)
53 sw $v0,4($v1)
54 lw $s0,24($fp)
55 lw $v1,52($fp)
56 lw $v0,48($fp)
57 mult $v1,$v0
58 mflo $v0
59 move $a0,$v0
60 li $a1,8 # 0x8
61 la $t9, calloc
62 jal $ra,$t9
63 sw $v0,8($s0)
64 lw $v0,24($fp)
```