Conjunto de instrucciones MIPS

66:20 Organizacin de Computadoras

Trabajo prctico 1

Axel Lijdens (95772)
Eduardo R Madariaga (90824)
Mauro Toscano (96890)

Univesidad de Buenos Aires - FIUBA

Contents

1 Objetivos

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

2 Alcance

Este trabajo prctico es de elaboracin grupal, evaluacin individual, y de carcter obligatorio para todos alumnos del curso.

3 Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya, la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4 Recursos

Usaremos el programa GXemul para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prcticos, una mquina MIPS corriendo una versin reciente del sistema operativo NetBSD. Durante la primera clase del curso presentaremos brevemente los pasos necesarios para la instalacin y configuracin del entorno de desarrollo.

5 Fecha de entrega

La ltima fecha de entrega y presentacin será el jueves 26 de abril de 2018.

6 Informe

El informe deber incluir:

- Documentacin relevante al diseo e implementacin del programa.

- Documentacin relevante al proceso de compilacin: cmo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba para los archivos matrix1, matrix2, y matrix3.
- Diagramas del stack de las funciones main y trasponer;
- El cdigo fuente, en lenguaje C.
- Este enunciado.

7 Implementación del programa

Para el diseño del programa se empezó por establecer los parámetros de entrada necesarios:

- Un archivo de entrada
- Un archivo de salida opcional. Si ninguno es provisto la salida será por stdout
- Opciones para mostrar un mensaje de ajuda y la versión del programa

Teniendo en cuenta lo propuesto se escribió el siguiente mensaje de ayuda:

```
1 Usage:
2    traspuesta -h
3    traspuesta -V
4    traspuesta [options] filename
5 Options:
6   -h, --help         Prints usage information.
7   -V, --version      Prints version information.
8   -o, --output       Path to output file.
```

Listing 1: Mensaje de ayuda del programa

Asimismo, se debieron tener en cuenta y validar los siguientes casos en donde algún factor es incorrecto y el mensaje devuelto por el probrama debe ser explicativo del error:

- **No se pudo abrir algun archivo:** se debe no solo expresar claramente que se falló al abrir un archivo, sino que tambien se debe especificar si fue en el de entrada o el de salida
- **formato inválido del archivo de entrada:** se refiere a una matriz de entrada que no tiene columnas o filas de igual largo, contiene algún caracter especial o no numérico, o cualquier divergencia del archivo esperado de entrada.

- **Archivos vacíos:** solo se aceptarán matrices de entrada con al menos una columma y una fila.

Una vez pasadas las validaciones, se procederá a cargar la matriz primero obteniendo la cantidad de filas/columnas (utilizando estos valores para allocar espacio en memoria que pueda albergar las matrices tanto de entrada como de salida) y luego cargando la matriz de entrada.

Se procede entonces a invocar a la función trasponer (que se encontrará tanto en assembler como en código c), pasandole los siguientes datos:

- *cantidad_de_filas*
- *cantidad_de_columnas*
- *matriz_entrada*
- *matriz_salida*

Finalmente se procede a guardar la matriz obtenida en el archivo de salida (proveido por el usuario o mostrada por stdout) y liberando la memoria que se encuentra alocada para albergar las matrices de entrada y salida.

8 Copmilación del programa

En el makefile, se especificaron las opciones para compilar el codigo. Es decir, se compilan los archivos **main.c** y **matrix.c** en el caso de compilar la totalidad del código en c.

make c

A la hora de compliar el código utilizando la función escrita en assembler, se utilizaron los archivos **main.c** y **matrix.S** con la opción **-D__MIPS__** para establecer las opciones de compilación. En este último caso se utilizo el entorno propuesto por la cátedra para compilar el programa y corer el programa en el sistema operativo NetBSD.

make assembly

9 Corridas de prueba

[insertar las imagenes de las corridas de pruebas]

10 Diagramas del Stack

[insertar diagramas del stack para la función trasponer]

11 Concluciones

[Incluir algun tipo de conclusiones]

11.1 Preguntas disparadoras

[Incluir algunas preguntas que hayan surgido de la elaboración del proyecto]

11.2 Problemas encontrados a lo largo del proyecto

[Incluir problemas que hayan surgido de la elaboración del proyecto]

11.3 Futuras investigaciones

[Incluir temas que podrían seguir estudiándose más a fondo]

12 Códigos fuente

12.1 MakeFile

```
1  c: main.c matrix.c
2    gcc main.c matrix.c -o tp
3
4  assembly: main.c matrix.S
5    gcc main.c matrix.S -o tp_assembly -D__MIPS__
```

Listing 2: makefile

12.2 Main

```
1  #include <ctype.h>
2  #include <getopt.h>
3  #include <inttypes.h>
4  #include <stdbool.h>
5  #include <stddef.h>
6  #include <stdint.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <unistd.h>
10 #include <string.h>
11 #include "matrix.h"
```

```
12
13 struct args {
14
15   /* Path del archivo con los datos de entrada */
16   const char* path_entrada;
17
18   /* Path salida */
19   const char* path_salida;
20
21   /* Boolean indica si se usa stdout */
22   bool usa_std_out;
23 };
24
25
26 /** Estructura que uitliza getopt_log para parsear
       los argumentos de linea de
27  * comandos. */
28 static const struct option _long_opts[] = {
29     {.name = "help", .has_arg = no_argument, .flag =
          NULL, .val = 'h'},
30     {.name = "version", .has_arg = no_argument, .
         flag = NULL, .val = 'V'},
31     {.name = "output", .has_arg = required_argument,
          .flag = NULL, .val = 'o'},
32     {0},
33 };
34
35 /**
36  * @brief Imprime un mensaje de ayuda y termina el
       programa.
37  *
38  * @param bin_name argv[0].
39  */
40 static void _print_help(const char *bin_name) {
41   printf("Usage:\n");
42   printf("   traspuesta -h\n");
43   printf("   traspuesta -V\n");
44   printf("   traspuesta [options] filename\n");
45   printf("Options:\n");
46   printf("  -h, --help        Prints usage
       information.\n");
47   printf("  -V, --version     Prints version
       information.\n");
48   printf("  -o, --output      Path to output file.\n
       ");
```

```
49 }
50
51 /**
52  * @brief Imprime la version del programa y termina.
53  *
54  * @param bin_name argv[0].
55  */
56 static void _print_version(const char *bin_name) {
57   printf("%s, version 1.00\n", bin_name);
58 }
59
60 static void _arg_parse(struct args* args,int argc,
       const char **argv) {
61
62   int ch = -1;
63   args->usa_std_out = true;
64
65   while ((ch = getopt_long(argc, (char **)argv, "hVo
       :", _long_opts, NULL)) != -1) {
66     switch (ch) {
67       case 'h':
68         _print_help(argv[0]);
69         exit(0);
70         break;
71
72       case 'V':
73         _print_version(argv[0]);
74         exit(0);
75         break;
76
77       case 'o':
78         args->usa_std_out = false;
79         args->path_salida = argv[optind - 1];
80         break;
81
82       /* this is returned when a required argument
            was not provided */
83       case '?':
84         exit(1);
85         break;
86
87     }
88   }
89
90   if(optind < argc){
```

```
91      args->path_entrada = argv[optind];
92      optind++;
93      }else{
94         fprintf( stderr, "No file specified\n");
95         exit(1);
96      }
97  }
98
99  void abrir_archivos(struct args args, FILE**
        archivo_entrada, FILE** archivo_salida){
100     *archivo_entrada = fopen(args.path_entrada, "r")
            ;
101     if (*archivo_entrada == 0) {
102        perror("Input file error");
103     }
104
105     if(args.usa_std_out)
106         *archivo_salida = stdout;
107     else{
108         *archivo_salida = fopen(args.path_salida, "w
               ");
109         if (*archivo_salida == 0) {
110             perror("Output file error");
111         }
112     }
113
114     //Si alguno de los dos fallo cancelo lo hecho y
            salgo
115     if(*archivo_entrada == 0 || *archivo_salida ==
            0){
116
117         if(*archivo_entrada != 0){
118             fclose(*archivo_entrada);
119         }
120
121         if(*archivo_salida != 0){
122             fclose(*archivo_salida);
123             remove(args.path_salida);
124         }
125
126         exit(1);
127     }
128 }
129
130 int main(int argc, const char *argv[]){
```

```
131     struct args args;
132     FILE* archivo_entrada;
133     FILE* archivo_salida;
134     int cantidad_de_filas;
135     int cantidad_de_columnas;
136     int cantidad_de_filas_traspuesta;
137     int cantidad_de_columnas_traspuesta;
138     long long int* matriz_entrada;
139     long long int* matriz_salida;
140     int indice_matriz_entrada = 0;
141     char* direccion_caracter_no_numerico = NULL;
142     char string_cargado[50];
143     long long int numero_cargado;
144     _arg_parse(&args,argc, argv);
145 
146     abrir_archivos(args, &archivo_entrada, &
            archivo_salida);
147 
148     fscanf(archivo_entrada, "%s ", string_cargado);
149     cantidad_de_filas = strtol(string_cargado, &
            direccion_caracter_no_numerico, 0);
150     if(*direccion_caracter_no_numerico != '\0'){
151         fprintf( stderr, "Input file doesn't have a
                valid format\n");
152         exit(1);
153     }
154     if(cantidad_de_filas < 1){
155         fprintf( stderr, "Number of rows must be
                positive\n");
156         exit(1);
157     }
158 
159 
160     fscanf(archivo_entrada, "%s ", string_cargado);
161     cantidad_de_columnas = strtol(string_cargado, &
            direccion_caracter_no_numerico, 0);
162     if(*direccion_caracter_no_numerico != '\0'){
163         fprintf( stderr, "Input file doesn't have a
                valid format\n");
164         exit(1);
165     }
166     if(cantidad_de_columnas < 1){
167         fprintf( stderr, "Number of columns must be
                positive\n");
168         exit(1);
```

```
169     }
170
171     matriz_entrada = malloc(sizeof(long long int) *
           cantidad_de_filas * cantidad_de_columnas);
172     matriz_salida = malloc(sizeof(long long int) *
           cantidad_de_filas * cantidad_de_columnas);
173
174     //Cargo la matriz de entrada
175     while(fscanf(archivo_entrada, "%s ",
           string_cargado) != EOF){
176
177         numero_cargado = strtoll(string_cargado, &
               direccion_caracter_no_numerico, 0); //
               Esta en base 10;
178
179         if(*direccion_caracter_no_numerico != '\0'){
180             fprintf( stderr, "Input file doesn't
                   have a valid format\n");
181             exit(1);
182         }
183
184         matriz_entrada[indice_matriz_entrada] =
               numero_cargado;
185         indice_matriz_entrada++;
186
187     }
188
189     trasponer(cantidad_de_filas,
           cantidad_de_columnas, matriz_entrada,
           matriz_salida);
190     cantidad_de_columnas_traspuesta =
           cantidad_de_filas;
191     cantidad_de_filas_traspuesta =
           cantidad_de_columnas;
192
193     fprintf(archivo_salida, "%u %u\n",
           cantidad_de_filas_traspuesta,
           cantidad_de_columnas_traspuesta );
194
195     unsigned i,j;
196     for (i = 0; i < cantidad_de_filas_traspuesta; i
           ++) {
197         for (j = 0; j <
               cantidad_de_columnas_traspuesta; j++) {
```

```
198             fprintf(archivo_salida, "%lld ",
                   matriz_salida[i *
                   cantidad_de_columnas_traspuesta + j])
                   ;
199         }
200         fputc('\n', archivo_salida);
201     }
202
203     free(matriz_entrada);
204     free(matriz_salida);
205
206 }
```

Listing 3: makefile

12.3 Matrix Method (c)

```
1  #include "matrix.h"
2
3  /**
4   * @brief Transpone una matriz almacenada en un
       array de long long.
5   *
6   * @param filas Cantidad de filas en la matriz.
7   * @param columnas Cantidad de columnas en la matriz
       .
8   * @param entrada Puntero a la matriz de entrada.
9   * @param salida Puntero a la matriz de salida.
10  * @return 0 siempre.
11  */
12 int trasponer(unsigned int filas, unsigned int
      columnas, const long long *entrada,
13               long long *salida) {
14   unsigned i,j;
15   for (i = 0; i < filas; i++) {
16     for (j = 0; j < columnas; j++) {
17       salida[j * filas + i] = entrada[i * columnas +
             j];
18     }
19   }
20
21   return 0;
22 }
```

Listing 4: Matrix method (c)

12.4 Matrix.h

```
1 #ifndef MATRIX_H
2 #define MATRIX_H
3
4 /** Transpone una matriz. */
5 int trasponer(unsigned int filas, unsigned int
     columnas, const long long *entrada,
6               long long *salida);
7
8 #endif
```

Listing 5: Matrix method (c)

12.5 Matrix Method (asm)

```
1  #include <mips/regdef.h>
2
3  # trasponer(unsigned int, unsigned int, long long
     const*, long long*):
4  .global trasponer
5  trasponer:
6      subu    sp, sp, 8       # muevo el stack pointer
7      sw      $fp, 4(sp)
8      sw      gp, 0(sp)
9
10     # parametros de entrada
11     sw        a0, 8(sp)      # filas
12     sw        a1, 12(sp)     # columnas
13     sw        a2, 16(sp)     # entrada
14     sw        a3, 20(sp)     # salida
15
16     # inicializa las variables locales
17     li        t0, 0          # $t0 = 0 (contador
         filas)
18     li        t2, 0          # $t2 = 0 (condicion de
          corte loop filas)
19
20 loop_rows:
21     slt     t2, t0, a0       # if( a0 <= t0 )
22     beq      t2, zero, end   #   goto end;
23
24     # inicializa el loop de columnas
25     li        t1, 0          # $t1 = 0 (contador
         columnas)
```

```
26      li        t3, 0          # $t3 = 0 (condicion de
            corte loop columnas)
27
28  loop_cols:
29      slt     t3, t1, a1          # if( a1 <= t1 )
30      beq     t3, zero, end_cols  #   goto end_cols;
31
32      # copia el valor de la matriz de entrada a la de
            salida
33      mulou   t4, t1, a0        # t4 = t1 * a0 + t0 (
            indice de salida)
34      addu    t4, t0
35      sll     t4, t4, 3         # multiplica por 8
36      addu    t4, a3
37
38      mulou   t5, t0, a1        # t5 = t0 * a1 + t1 (
            indice de entrada)
39      addu    t5, t1
40      sll     t5, t5, 3
41      addu    t5, a2
42
43      lw      t6, 0(t5)         # carga el valor del
            indice
44      lw      t7, 4(t5)         # en 2 partes (por ser
            un long long)
45
46      sw        t6, 0(t4)       # copia las 2 words
47      sw        t7, 4(t4)       # copia las 2 words
48
49      # incrementa el contador de columas
50      addi    t1, t1, 1         # t1 = t1 + 1
51      j        loop_cols        # jump a loop_cols
52
53  end_cols:
54      # incrementa el contador de filas
55      addi    t0, t0, 1         # t0 = t0 + 1
56      j        loop_rows        # jump a loop_cols
57
58  end:
59      # restaura los resgitros
60      lw        gp, 0(sp)
61      lw        $fp, 4(sp)
62      li        v0, 0         # v0 = 0
63      addiu     sp, sp, 8     # libera el stack
64      jr        ra            # retorna
```

Listing 6: matrix method (asm