UBA FACULTAD DE INGENIERÍA

66.20 Organización de Computadoras

Trabajo Práctico 1

2^{do} Cuatrimestre 2017

Integrantes:

Rodriquez Longhi, Federico 93336
federico.rlonghi@gmail.com
Deciancio, Nicolás Andrés 92150
nicodec_89@hotmail.com

Índice

1. Introducción

Este trabajo práctico, consistió principalmente en la codificación de la función palíndromo en código mips, implementada previamente en el tp0 en lenguaje C. Además, se reestructuro el método de input/output del programa para poder limitar la cantidad de acceso a escritura y lectura. Esto se logró con la utilización de buffers, tanto de escritura como de lectura. Los cuales se leen/escriben bien cuando se llenan o cuando se llego al final de la entrada y no hay que analizar más palabras.

Con esto tomamos como objetivo también poder hacer un análisis del costo de las llamadas a sistema. La idea es ver cuanto pierde un programa en performance al realizar estas llamadas.

Todo esto, nos obligó a interiorizarnos mucho más con el conjunto de instrucciones assembly para mips 32, y el uso de syscalls. Además, vimos de forma práctica, como llamar a funciones de archivos .s desde codigo C. El uso de un debugger fue de gran importancia para todo el desarrollo del prorgama en mips. A continuación, se ve la documentación del programa, forma de compilarlo, casos de prueba armados, análisis del problema y su solución, y el código fuente del programa.

2. Documentación

El uso del programa se compone de las siguientes opciones que le son pasadas por parámetro:

- `-h` o `--help`: muestra la ayuda.
- `-V` o `--version`: muestra la versión.
- `-i` o `--input`: recibe como parámetro un archivo de texto como entrada. En caso de que no usar esta opción, se toma como entrada la entrada estándar.
- `-o` o `--output`: recibe como parámetro un archivo de texto como salida. En caso de que no usar esta opción, se toma como salida la salida estándar.
- `-I` o `--ibuf-bytes`: recibe como parámetro el tamaño en bytes del buffer de entrada.
- `-O` o `--iobuf-bytes`: recibe como parámetro el tamaño en bytes del buffer de salida.

2.1. Diagrama de Proceso

A continuación se muestra un diagrama del proceso que recorre el programa, haciendo mención de los buffers que se utilizan y las acciones de comunicación de datos entre ellos.

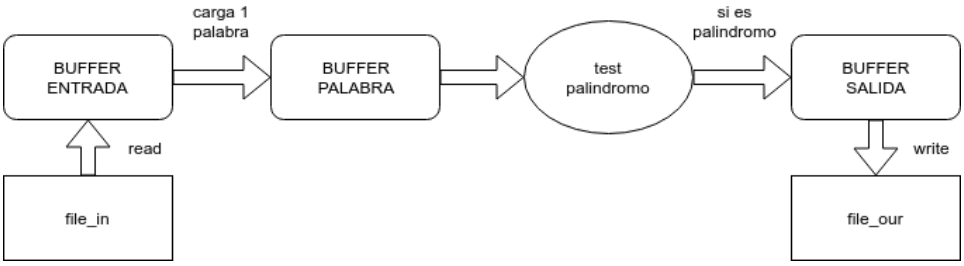

Figura 1: Diagrama del proceso de comunicación entre buffers

3. Compilación

Dentro del directorio raiz se encuentra un Makefile. Ejecuntando `make` se compilara el programa y se generará el ejecutable `tp1`.

4. Pruebas

Para las pruebas se proveen de dos scripts que las ejecutan. Se utilizaron las mismas pruebas que para el tp0, solo que fueron corridas varias veces variando los parametros -O y -I del programa.

El primer script `test.sh` ejecuta los ejemplos del enunciado.

El segundo script `test_p.sh` ejecuta las pruebas propias. Este archivo esta diseñado para poder agregar pruebas de forma sencilla, simplemente se debe agregar una linea en el sector de pruebas de la siguiente manera:

`make_test <nombre><entrada de texto><salida esperada>`

Este script crea los archivos correspondientes en la carpeta tests (dentro del directorio sobre el cual se ejecuta). Los archivos creados son de la forma:

- `test-<nombre del test>_in`: archivo de entrada
- `test-<nombre del test>_out`: archivo de salida generado por el programa
- `test-<nombre del test>_expected`: archivo de salida esperado

4.1. Corridas de Prueba

A continuación se muestran las corridas de prueba generadas por el script:

```
1
2  Compiling Source
3  Compilation Success
4  Starting Tests
5
6  Test: one_letter_a
7  Test passed
8
9  Test: empty_file
10 Test passed
11
12 Test: no_palindroms
13 Test passed
14
15 Test: todos_palindromos
16 Test passed
17
18 Test: varias_lineas
19 Test passed
20
21 Test: all_letters
22 Test passed
23
24 Test: case_sensitive
25 Test passed
26
27 Test: numbers_and_letters
28 Test passed
29
30 Test: text_with_dash
31 Test passed
32
33 --------------------
34 All 9 tests passed!!!
35 --------------------
```

5. Análisis

Hemos utilizado este programa para analizar los tiempos de las llamadas al kernel. Las syscalls son llamadas al sistema, lo que requiere un cambio de entorno (de modo usuario a modo kernel), y esto consume mayor tiempo que si no se hiciese un cambio de entorno.

Como se provee un buffer de entrada y de salida al modificar el tamaño de estos, estamos disminuyendo o aumentando los syscalls read y write, es decir, cuanto mayor el tamaño del buffer menor cantidad de syscalls y

viceversa.

Lo que esperamos obtener como resultado es que al tener un buffer pequeño (por ejemplo de 1 byte) la cantidad de syscalls (la cantidad de syscalls read va a ser la cantidad de letras(bytes) que tenga el archivo) sea mucha, y por lo tanto obtendremos un tiempo de ejecución mayor que el tiempo de ejecucion del mismo programa pero con un buffer mayor (por ejemplo de 1000 bytes, si el archivo es menor o igual al tamaño del buffer haremos un solo read). El costo de tener un buffer de mayor tamaño es el espacio utilizado por el programa. Esta claro que cuanto mayor sea el buffer mayor sera la memoria requerida por el programa.

Para realizar este análisis utilizamos un archivo de texto como entrada de un tamaño de 317.520 bytes, el cual consiste de muchos palindromos para que el programa utilice la syscall de write. Luego medimos los tiempos de ejecucion del programa. Para ello se provee con un script llamado `time_analisis`. A continuación se muestra una tabla con los resultados:

Tiempos de ejecucion

Tamaño Buffer Entrada	Tamaño Buffer Salida	Tiempo Ejecucion
1	1	15.918s
10	10	2.352s
100	100	0.566s
1000	1000	0.402s
10	1	10.348s
100	1	9.531s
1000	1	9.375s
1	10	7.699s
1	100	6.652s
1	1000	6.531s

Se ve claramente como las ejecuciones con buffers mas grandes tardan un tiempo considerablemente menor al que los que tienen buffers de menor tamaño. Siendo el más lento el de buffers de tamaño 1 y 1 y el más rápido el de tamaño 1000 y 1000.

Por lo tanto el analisis cumple con lo que predijimos anteriormente.

6. Conclusión

De este trabajo práctico nos llevamos principalmente que es muy importante la experiencia y conocimiento de las instrucciones de Assembly para mips 32 y como usarlas eficientemente. También, pudimos ver por nuestros propios ojos como puede llegar a afectar al tiempo de ejecución de un programa la cantidad de syscalls que se realizan. Acotando las syscalls de read y de write con la utilización de un buffer de escritura y otro de lectura, vimos un speed up global del programa. Como dijimos previamente, cada

syscall implica un cambio de entorno de modo usuario a modo kernel. Y si se repite consistemente este 'switch', el tiempo de ejecución del programa se ve fuertemente afectado. Por esto, siempre que se pueda acotar la cantidad de llamadas al sistema de un programa, se debería tratar de optimizarlas. En este caso particular, se acotó esa cantidad usando los buffers. Fue muy importante también el uso de un debugger para poder arreglar problemas durante la codificación.

Apéndice

A. Stacks de Funciones

A continuación se muestran como fueron conformados los stacks de las funciones programadas en MIPS segun lo dispuesto por la ABI.

A.1. Stack palindromo

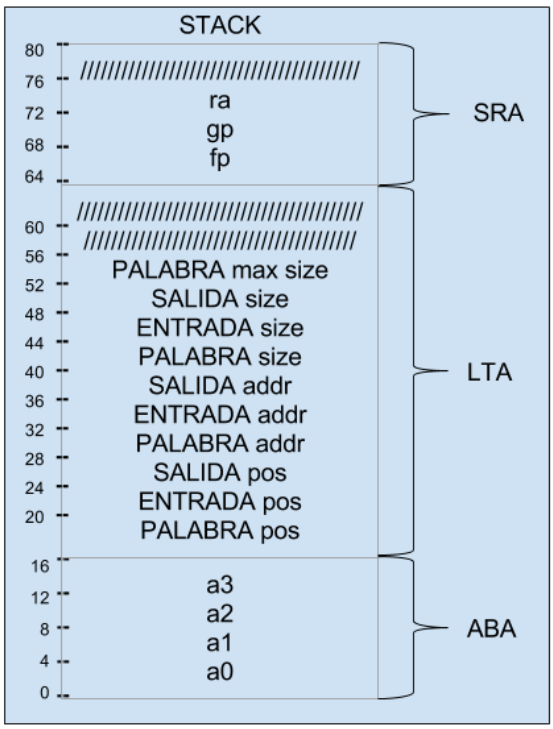

Figura 2: Stack de la función `int palindromo(int, size_t, int, size_t)`

A.2. Stack testpal

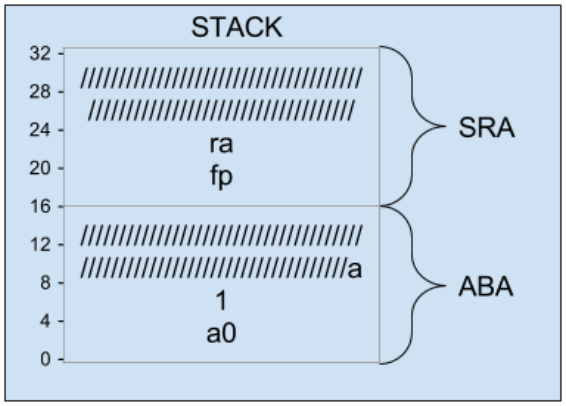

Figura 3: Stack de la función `int testpal(char *, size_t)`

B. Código en C

El siguiente código corresponde al cuerpo principal del programa. Se encarga de procesar los parámetros, abrir los archivos, llamar a la función palíndromo (escrita en MIPS) y luego cerrar los archivos correspondientes.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <unistd.h>
5  #include <getopt.h>
6  #include <errno.h>
7  #include <string.h>
8
9  extern size_t mystrlen(const char*);
10 extern int palindrome(int, size_t, int, size_t);
11
12 /* imprimir el uso de tp0 */
13 void print_usage() {
14     printf("Usage: tp0 -i [input_file] -o [output_file]\n");
15 }
16
17 /* imprimir la pagina de ayuda */
18 void print_help() {
19     printf("\tUsage:\n"
20       "\t\ttp0 -h\n"
21       "\t\ttp0 -V\n"
22       "\t\ttp0 [options]\n"
23       "\tOptions:\n"
24       "\t\t-V, --version\tPrint version and quit.\n"
25       "\t\t-h, --help\tPrint this information.\n"
26       "\t\t-i, --input\tLocation of the input file.\n"
27       "\t\t-o, --output\tLocation of the output file.\n"
```

```
28          "\t\t-I, --ibuf-bytes\tByte count of the input buffer.\
                n"
29          "\t\t-O, --obuf-bytes\tByte count of the output buffer
                .\n"
30        "\tExamples:\n"
31        "\t\ttp0 -i ~/input -o ~/output\n");
32  }
33
34  /* imprimir la version del programa */
35  void print_version(){
36      printf("tp1 1.0\n");
37  }
38
39  int main(int argc, char *argv[]) {
40
41      int opt= 0;
42
43      int help = -1;
44      int version = -1;
45      int input = -1;
46      int output = -1;
47      int ibuf = 1;
48      int obuf = 1;
49
50      char *input_filename = NULL;
51      char *output_filename = NULL;
52
53      // especificacion de las opciones
54      static struct option long_options[] = {
55          {"help",          no_argument,       0,  'h' },
56          {"version",       no_argument,       0,  'V' },
57          {"input",         required_argument, 0,  'i' },
58          {"output",        required_argument, 0,  'o' },
59          {"ibuf-bytes",    required_argument, 0,  'I' },
60          {"obuf-bytes",    required_argument, 0,  'O' },
61          {0,               0,                 0,  0   }
62      };
63
64      int long_index = 0;
65
66      // evaluacion de los parametros enviados al programa
67      while ((opt = getopt_long(argc, argv,"hVui:o:I:O:",
68                      long_options, &long_index )) != -1) {
69          switch (opt) {
70              case 'h' :
71                  help = 0;
72                  break;
73              case 'V' :
74                  version = 0;
75                  break;
76              case 'i' :
77                  input = 0;
78                  input_filename = optarg;
79                  break;
```

```
80              case 'o' :
81                  output = 0;
82                  output_filename = optarg;
83                  break;
84              case 'I':
85                  ibuf = atoi(optarg);
86                  break;
87              case 'O':
88                  obuf = atoi(optarg);
89                  break;
90              case '?':
91                exit(1);
92              default:
93                  print_usage();
94                  exit(EXIT_FAILURE);
95          }
96      }
97
98      // procesamiento de los parametros
99      if (help == 0) {
100         print_help();
101         exit(0);
102     }
103     else if (version == 0) {
104         print_version();
105         exit(0);
106     }
107
108     /* Si no se recibe parametro de ayuda o version se ejecuta
            el programa */
109
110     // estableciendo los archivos de entrada y salida
111     FILE *input_file = stdin;
112     FILE *output_file = stdout;
113
114     if (input == 0){
115         input_file = fopen(input_filename,"r");
116         if (input_file == NULL) {
117             printf ("can't open input file, errno = %d\n",
                        errno);
118             return 1;
119         }
120     }
121     if (output == 0){
122         output_file = fopen(output_filename,"w");
123         if (output_file == NULL) {
124             printf ("Can't open output file, errno = %d\n",
                        errno);
125             return 1;
126         }
127     }
128
129     /* ejecucion del programa */
130
```

```
131     int file_in = fileno(input_file);
132     int file_out = fileno(output_file);
133 
134     int a = palindrome(file_in,ibuf,file_out,obuf);
135 
136     if (input == 0){
137         fclose(input_file);
138     }
139     if (output == 0){
140         fclose(output_file);
141     }
142 
143     return 0;
144 }
```

C. Función Palíndromo en MIPS

Esta función escrita en MIPS es la que corresponde a la pedida en el enunciado, en ella se llama a mymalloc y a testpal (descripta en la próxima sección).

```
1  # palindrome.S - ver tp1.c.
2  #
3  # $Date: 2017/09/24 17:12:06 $
4  
5  #include <sys/syscall.h>
6  #include <mips/regdef.h>
7  
8    .text
9    .align  2
10 
11   .globl  palindrome
12   .ent    palindrome
13 
14 palindrome:
15   .frame $fp, 88, ra
16   .set    noreorder
17   .cpload t9
18   .set    reorder
19 
20   subu    sp, sp, 88
21 
22   .cprestore 60
23 
24   sw  ra,  72(sp)
25   sw  gp,  68(sp)
26   sw  $fp, 64(sp)
27 
28   sw  a0, 0(sp)       #file_in
29   sw  a1, 4(sp)       #ibuf
30   sw  a2, 8(sp)       #file_out
31   sw  a3, 12(sp)      #obuf
```

```
32
33    move    $fp, sp
34
35    ####################################################
36    #cargo los buffers necesarios para operar
37    #buffer_entrada -> empty
38    #buffer_word -> empty
39    #bufer_salida -> empty
40
41    li  a0, 100         # cargo el parametro para mymalloc
42    sw  a0, 52(sp)      # guardo en el stack el tamaño maximo del
          buffer palabra
43    jal mymalloc        # llamo a la funcion
44    sw  v0, 28(sp)      # guardo la direccion de memoria
          reservada para el buffer de PALABRA
45    add t0, zero, zero
46    sw  t0, 40(sp)      # guardo el tamaño del buffer (
          incialmente vacio)
47
48    lw  a0, 4(sp)       # cargo el parametro para mymalloc
49    jal mymalloc        # llamo a la funcion
50    sw  v0, 32(sp)      # guardo la direccion de memoria
          reservada para el buffer de ENTRADA
51    add t0, zero, zero
52    sw  t0, 44(sp)      # guardo el tamaño del buffer (
          incialmente vacio)
53
54    lw  a0, 12(sp)      # cargo el parametro para mymalloc
55    jal mymalloc        # llamo a la funcion
56    sw  v0, 36(sp)      # guardo la direccion de memoria
          reservada para el buffer de SALIDA
57    add t0, zero, zero
58    sw  t0, 48(sp)      # guardo el tamaño del buffer (
          incialmente vacio)
59
60    sw  zero, 84(sp)  # marca fin de proceso
61    add t0, zero, zero
62    sw  t0, 16(sp)      # guardo SALIDA pos = 0
63    sw  t0, 20(sp)      # guardo ENTRADA pos = 0
64    sw  t0, 24(sp)      # guardo PALABRA pos = 0
65
66    ####################################################
67
68  leerArchivo:
69    #read file_in -> buffer_entrada (I Bytes)
70
71    li    v0, SYS_read    # ver dentro de <sys/syscall.h>.
72    lw    a0, 0(sp)       # a0: file descriptor number.
73    lw    a1, 32(sp)      # a1: data pointer.
74    lw    a2, 4(sp)       # a2: available space.
75    syscall
76
77    bne   a3, zero, read_error    #verifico error de lectura
78
```

```
79    seq   t0, v0, zero
80    sw    t0, 84(sp)    # si no se leyeron mas caracteres pongo
          el flag en 1
81    beqz  v0, escribirArchivo # si read no lee nada finalizo el
          programa
82
83    add   t0, zero, zero  # en t0 cargo el indice inicial del
          buffer
84    sw    t0, 20(sp)      # guardo en el stack ENTRADA pos
85
86    sw    v0, 44(sp)      # guardo en el stack ENTRADA size
87
88  leoByte:
89    lw  t5, 44(sp)      # cargo en t5 ENTRADA size
90    beq t5, zero, leerArchivo # si ENTRADA size == 0 =>
          leerArchivo
91
92    # c = leo un byte <- buffer_entrada
93    lw  t0, 20(sp)      # cargo en t0 ENTRADA pos (t0 = i)
94    lw  t1, 32(sp)      # cargo en t1 ENTRADA addr
95    add t1, t1, t0      # sumo ENTRDA addr + pos
96    lb  t2, 0(t1)       # c: t2 = ENTRADA[i]
97
98    sub t5, t5, 1       # ENTRADA size - 1
99    add t0, t0, 1       # ENTRADA pos + 1
100   sw  t5, 44(sp)      # guardo en stack ENTRADA size
101   sw  t0, 20(sp)      # guardo en stack ENTRADA pos
102
103 testAlfaNumerico:
104   # si c no es alfanumerico tengo que testear
105   # si lo que hay en el buffer de word es palindromo
106   slti    t3, t2, 48
107   sne     t4, t2, 45
108   add     t3, t4, t3
109   beq     t3, 2, test_palindromo
110   sgt     t3, t2, 57
111   slti    t4, t2, 65
112   add     t3, t3, t4
113   beq     t3, 2, test_palindromo
114   sgt     t3, t2, 90
115   slti    t4, t2, 97
116   add     t3, t3, t4
117   sne     t4, t2, 95
118   add     t3, t3, t4
119   beq     t3, 3, test_palindromo
120   slti    t3, t2, 123
121   beqz    t3, test_palindromo
122
123   # sino: pongo a c en el buffer de palabra (voy construyendo
          la palabra)
124   lw  t7, 28(sp)      # cargo en t7 PALABRA addr
125   lw  t8, 16(sp)      # cargo en t8 PALABRA pos (t8 = j)
126   add t7, t7, t8      # t7 cargo PALABRA addr + pos
127   sb  t2, 0(t7)       # PALABRA[j] = c
```

```
128
129   lw  t0, 40(sp)      # cargo en t0 PALABRA size
130   add t0, t0, 1       # PALABRA size + 1
131   sw  t0, 40(sp)      # guardo en el stack PALABRA size
132
133   addi    t8, 1       # PALABRA pos + 1
134   sw      t8, 16(sp)  # guardo PALABRA pos en el stack
135
136     # (ver que el buffer word no se llene)
137   # (si se llena pedir mas memoria, hacer swap con memoria
        pedida y memoria vieja)
138   lw    t6, 52(sp)  # tamaño maximo de palabra
139   sne   t6, t8, t6  # si no son iguales t6 = 1 sino t6 = 0
140   sub   t6, t6, 1
141   bltzal  t6, reasignar_buffer_palabra
142
143
144
145   b leoByte # vuelvo a leer un byte
146
147 reasignar_buffer_palabra:
148   sw    ra, 80(sp)
149   lw    a0, 28(sp)
150   lw    a1, 52(sp)
151   mul   a2, a1, 2
152   jal   myremalloc
153   sw    v0, 28(sp)
154   sw    a2, 52(sp)
155
156   lw    ra, 80(sp)
157   sw    a2, 80(sp)
158   j   ra
159
160 test_palindromo:
161
162   #me fijo si buffer_word es palindromo
163
164   lw  a0, 28(sp)      # cargo en a0 la PALABRA addr
165   lw  a1, 40(sp)      # cargo en a1 PALABRA size
166   jal testpal         # esPalindromo(PALABRA addr, PALABRA size
        )
167
168   beqz v0, esPalindromoTrue # si v0 = 0 => palindromo true
169
170   # si no es palindromo vacio el buffer PALABRA
171 noPalindromo:
172   move    t0, zero  # t0 = 0
173   sw      zero, 16(sp)      # PALABRA pos = 0
174   sw      zero, 40(sp)
175   b       leoByte
176
177 esPalindromoTrue:
178   # buffer_word -> buffer_salida
179   # (hay que fijarse que buffer salida tenga espacio suficiente
```

```
    )
  # si se llena buffer salida => escribirArchivo
  # buffer_salida + '\n'
  # buffer_salida.size ++

  lw      t2, 40(sp)  # cargo en t2 PALABRA size
  #lw  t3, 24(sp)  # cargo en t3 SALIDA pos
  add     t4, zero, zero # en t4 posicion caracter PALABRA (i =
      0)
  sw    zero, 56(sp)
  sw      zero, 76(sp)

llenarBufferSalida:
  lw      t0, 28(sp)      # cargo en t0 PALABRA addr
  lw      t1, 36(sp)      # cargo en t1 SALIDA addr
  lw      t3, 24(sp)      # cargo en t3 SALIDA pos
  lw    t4, 56(sp)

  addu    t1, t1, t3      # en t1 SALIDA addr + pos
  add     t0, t0, t4      # en t0 PALABRA addr + pos
  lb      t5, 0(t0)       # t5 = PALABRA[i]
  sb      t5, 0(t1)       # SALIDA[j] = PALABRA[i]
  addi    t3, t3, 1       # j + 1
  addi    t4, t4, 1       # i + 1
  sw      t3, 24(sp)      # guardo SALIDA pos en el stack
  sw    t4, 56(sp)

  lw      t6, 12(sp)              # cargo en t6 SALIDA max size

  beq     t6, t3, escribirArchivo # si se llena el buffer
      escribo

  # me fijo si no se termino la palabra
  blt t4, t2, llenarBufferSalida  # PALABRA pos < PALABRA size
      => sigo llenando

write_endofline:
  # sino => se termino la palabra -> guardo \n en buffer salida
  lw      t6, 12(sp)              # cargo en t6 SALIDA max size
  lw      t3, 24(sp)      # cargo en t3 SALIDA pos
  add     t7, zero, 10    # guardo en t7 el valor 10 ('\n')
  lw      t1, 36(sp)      # cargo en t1 SALIDA addr
  add     t1, t1, t3      # en t1 SALIDA addr + pos
  sb      t7, 0(t1)       # SALIDA[j] = '\n'
  addi    t3, t3, 1       # j + 1
  sw      t3, 24(sp)      # guardo SALIDA pos en el stack
  add     t8, zero, 1
  sw      t8, 76(sp)
  add     t4, t4, 1
  sw      zero, 40(sp)

  beq     t3, t6, escribirArchivo # si se llena el buffer
      escribo (ojo con los indices)

```

```
229   b leoByte
230
231
232 escribirArchivo:
233   # write buffer_salida -> file_out
234   sw      t2, 40(sp)  # cargo en t2 PALABRA size
235   sw      t4, 56(sp)
236
237   li    v0, SYS_write   # write
238   lw    a0, 8(sp)       # cargo en a0 el fno del archivo de
          salida
239   lw    a1, 36(sp)      # cargo en a1 la direccion del buffer
          de salida
240   lw    a2, 24(sp)      # cargo en a2 el tamaño del buffer
241   syscall
242
243   bne   a3, zero, write_error    #verifico error de escritura
244
245   lw    t2, 84(sp)    # busco la marca de fin de procesamiento
246   bnez  t2, palindromoReturn  # si la marca esta en 1 salgo del
          programa
247
248   add   t0, zero, zero
249   sw    t0, 24(sp)      # guardo en stack SALIDA pos = 0
250
251   lw      t2, 40(sp)  # cargo en t2 PALABRA size
252   lw      t4, 56(sp)
253   beq     t4, t2, write_endofline
254
255   lw    t8, 76(sp)
256   #beq  t8, zero, llenarBufferSalida
257   blt t4, t2, llenarBufferSalida
258
259   #sw  t0, 48(sp)      # guardo en stack SALIDA size = 0
260
261   b leoByte           #vuelvo a intentar leer bytes del buffer
          de entrada
262
263 write_error:
264 read_error:
265   jal     free_memory
266
267   li  v0, SYS_exit
268   li  a0, 1
269   syscall
270
271 free_memory:
272   sw    ra, 80(sp)
273   lw      a0, 28(sp)
274   jal     myfree
275   lw      a0, 32(sp)
276   jal     myfree
277   lw      a0, 36(sp)
278   jal     myfree
```

```
279   lw    ra, 80(sp)
280   j       ra
281
282 palindromoReturn:
283   jal     free_memory
284
285   move    v0, zero
286   move  sp, $fp
287
288   lw  ra,  72(sp)
289   lw  gp,  68(sp)
290   lw  $fp, 64(sp)
291
292   addu  sp, sp, 88
293   j     ra
294   .end  palindrome
```

D. Función testpal

Hemos desarrollado una función testpal.S la cual recibe en a0 la dirección de la palabra a evaluar, en a1 la longitud de la palabra y devuelve en v0: 0 si la palabra es palíndromo o 1 si no lo es.

```
1  #include <sys/syscall.h>
2  #include <mips/regdef.h>
3
4    .text
5    .align  2
6
7    .globl  testpal
8    .ent  testpal
9  testpal:
10     .frame $fp, 36, ra
11   .set  noreorder
12   .cpload t9
13   .set  reorder
14
15     subu  sp, sp, 36
16
17   .cprestore 24
18
19   sw  ra,  20(sp)
20   sw  $fp, 16(sp)
21     sw  a0,  0(sp)
22   sw  a1, 4(sp)
23     sw  t4, 28(sp)          # no estoy seguro de si se usan
         previamente por lo tanto los guardo para prevenir
         sobreescritura
24     sw  t5, 32(sp)
25
26     move  $fp, sp
27
```

```
28      move    t4, a1
29      add     t5, zero, zero # i = 0
30
31  test:
32      slti  t0, a1, 2           # si a1 (len palabra) < 2 => es
            palindromo
33    bne     t0, zero, esTrue
34
35      #add     a0, a0, t5          # palabra[i]
36      lb      t1, 0(a0)           # t1 = palabra[i]
37
38      move    t8, t1
39      jal     to_lowercase        # lo paso a lowercase
40      move    t1, t8              # t1 = tolower(palabra[i])
41
42      sub     t2, a1, 1          # t2 = len - 1
43      add     t2, t2, a0
44      lb      t2, 0(t2)           # t2 = palabra[len-1]
45
46      move    t8, t2
47      jal     to_lowercase
48      move    t2, t8              # t2 = tolower(palabra[len -
            1])
49
50      bne     t1, t2, esFalse     # si t1 != t2 no es palindromo
51
52      #sino actualizo los indices
53
54      sub     t4, t4, 1
55      add     a0, a0, 1           # muevo el puntero de t8 a la
            siguiente letra
56      sub     a1, a1, 2           # len = len - 2
57      addi    t5, t5, 1           # i = i + 1
58
59      blt     t4, t5, esTrue
60      j test
61
62  esFalse:
63      # si es false devuelvo 1
64      addi v0, zero, 1
65      j return
66
67  esTrue:
68      # si es true devuelvo 0
69      add v0, zero, zero
70      j return
71
72  to_lowercase:
73    sgt   t7, t8, 64
74    slti  t6, t8, 91
75    add   t6, t7, t6
76    beq   t6, 2, is_upper
77    #move   v0, t8
78    j     ra
```

```
79
80 is_upper:
81   add    t8, t8, 32
82   j      ra
83
84 return:
85   move   sp, $fp
86   lw       ra, 20(sp)
87   lw       $fp, 16(sp)
88     lw       t4, 28(sp)
89     lw       t5, 32(sp)
90   addu   sp, sp, 36
91   j      ra
92   .end   testpal
```

E. Función testpal en C

La siguiente función es el equivalente en C de testpal.S

```
1  int testpal(char * palabra, int len)
2  {
3      if (len < 2) {
4      return true;
5    }
6
7      char primero = tolower(palabra[0])
8      char ultimo = tolower(palabra[len - 1])
9
10     if (primero == ultimo)
11   {
12     return testpal(palabra + 1, len -2);
13   }
14
15   return false;
16 }
```

F. Enunciado

El enunciado se encuentra anexado al final de este documento.