

# Trabajo Práctico Nro. 1: programación MIPS

Lucas Verón, *Padrón Nro. 89.341*  
lucasveron86@gmail.com

Eliana Diaz, *Padrón Nro. 89.324*  
diazeliana09@gmail.com

Alan Helouani, *Padrón Nro. 90.289*  
alanhelouani@gmail.com

2do. Cuatrimestre de 2017  
66.20 Organización de Computadoras – Práctica Martes  
Facultad de Ingeniería, Universidad de Buenos Aires

## Resumen

El presente proyecto tiene por finalidad familiarizarnos con el conjunto de instrucciones MIPS y el concepto de ABI

## 1. Introducción

Se detallará el diseño e implementación de un programa en lenguaje C y MIPS que procesa archivos de texto por línea de comando, como así también la forma de ejecución del mismo y los resultados obtenidos en las distintas pruebas ejecutadas.

El programa recibe los archivos o streams de entrada y salida, e imprime aquellas palabras del archivo de entrada (componentes léxicos) que sean palíndromos.

Se define como palabra a aquellos componentes léxicos del stream de entrada compuestos exclusivamente por combinaciones de caracteres a-z, 0-9, - (signo menos) y (*guiónbajo*).

Por otro lado, se considera que una palabra, número o frase, es *palíndroma* cuando se lee igual hacía adelante que hacía atrás.

Se implementará una función "palindrome" la cual se encargará de verificar si efectivamente la palabra es o no palíndroma. La función estará escrita en assembly MIPS.

Los streams serán leídos y escritos de a bloques de memoria configurables, los cuales serán almacenados en un "buffer" para luego ser leídos de a uno.

## 2. Diseño

Las funcionalidades requeridas son las siguientes:

- Ayuda (Help): Presentación un detalle de los comandos que se pueden ejecutar.
- Versión: Se debe indicar la versión del programa.
- Procesar los datos:
  - Con especificación sólo del archivo de entrada.
  - Con especificación sólo del archivo de salida.
  - Con especificación del archivo de entrada y de salida.
  - Sin especificación del archivo de entrada ni de salida.
- Setting del tamaño del buffer in y buffer out; indicando de a cuantos caracteres se debe leer y escribir.

En base a estas funcionalidades, se modularizó el código a fin de poder reutilizarlo y a su vez que cada método se encargue de ejecutar una única funcionalidad.

## 3. Implementación

### 3.1. Código fuente en lenguaje C: tp1.c

```
1  /*
2  =====
3  Name      : tp1.c
4  Author    : Grupo orga 66.20
5  Version   : 1
6  Copyright : Orga6620 - Tp1
7  Description : Trabajo practico 1: Programacion MIPS
8  =====
9
10 */
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include <getopt.h>
15 #include <unistd.h>
16 #include "process.h"
17
18 #define VERSION "1.1"
19
20 #define FALSE 0
21 #define TRUE 1
22
23 size_t ibytes = 1;
```

```

24 size_t obytes = 1;
25
26 enum ParameterState {
27     OKEY = 0, INCORRECT_QUANTITY_PARAMS = 1,
28     INCORRECT_MENU = 2, ERROR_FILE = 3, ERROR_BYTES
29     = 6
30 };
31
32 int executeHelp() {
33     fprintf(stdout, "Usage: \n");
34     fprintf(stdout, "    tp1 -h \n");
35     fprintf(stdout, "    tp1 -V \n");
36     fprintf(stdout, "    tp1 [options] \n");
37     fprintf(stdout, "Options: \n");
38     fprintf(stdout, "    -V, --version
39         Print version and quit. \n");
40     fprintf(stdout, "    -h, --help
41         Print this information. \n");
42     fprintf(stdout, "    -i, --input
43         Location of the input file. \n");
44     fprintf(stdout, "    -o, --output
45         Location of the output file. \n");
46     fprintf(stdout, "    -I, --ibuf-bytes      Byte
47         -count of the input buffer. \n");
48     fprintf(stdout, "    -O, --obuf-bytes      Byte
49         -count of the output buffer. \n");
50     fprintf(stdout, "Examples: \n");
51     fprintf(stdout, "    tp1 -i ~/input -o ~/output \
52         n");
53
54     return OKEY;
55 }
56
57 int executeVersion() {
58     fprintf(stdout, "Version: \"%s\" \n", VERSION);
59
60     return OKEY;
61 }
62
63 int executeByMenu(int argc, char **argv) {
64     int inputFileDefault = FALSE;
65     int outputFileDefault = FALSE;
66     FILE * fileInput = stdin;
67     FILE * fileOutput = stdout;
68
69     // Always begins with /
70     if (argc == 1) {
71         // Run with default parameters
72         inputFileDefault = TRUE;
73         outputFileDefault = TRUE;
74     }
75
76     char * pathInput = NULL;
77     char * pathOutput = NULL;

```

```

69     char * iBufBytes = NULL;
70     char * oBufBytes = NULL;
71
72     /* Una cadena que lista las opciones cortas validas
73        */
74     const char* const smallOptions = "Vhi:o:I:O:";
75
76     /* Una estructura de varios arrays describiendo los
77        valores largos */
78     const struct option longOptions[] = {
79         {"version",          no_argument,
80          0, 'V' },
81         {"help",            no_argument,
82          0, 'h' },
83         {"input",           required_argument,
84          0, 'i' }, // optional_argument
85         {"output",          required_argument,
86          0, 'o' },
87         {"ibuf-bytes",      required_argument, 0,
88          'I' },
89         {"obuf-bytes",      required_argument, 0,
90          'O' },
91         {0,                  0,
92          0, 0 }
93     };
94
95     int incorrectOption = FALSE;
96     int finish = FALSE;
97     int result = OKEY;
98     int longIndex = 0;
99     char opt = 0;
100
101     while ((opt = getopt_long(argc, argv, smallOptions,
102                             longOptions, &longIndex )
103            ) != -1 &&
104            incorrectOption ==
105            FALSE && finish ==
106            FALSE) {
107
108         switch (opt) {
109             case 'V' :
110                 result = executeVersion();
111                 finish = TRUE;
112                 break;
113             case 'h' :
114                 result = executeHelp();
115                 finish = TRUE;
116                 break;
117             case 'i' :
118                 pathInput = optarg;
119                 break;
120             case 'o' :
121                 pathOutput = optarg;
122                 break;
123             case 'I' :

```

```

110         iBufBytes = optarg;
111         break;
112     case 'O' :
113         oBufBytes = optarg;
114         break;
115     default:
116         incorrectOption = TRUE;
117     }
118 }
119
120 if (incorrectOption == TRUE) {
121     fprintf(stderr, "[Error] Incorrecta option
122     de menu.\n");
123     return INCORRECT_MENU;
124 }
125
126 if (finish == TRUE) {
127     return result;
128 }
129
130 if (iBufBytes != NULL) {
131     char *finalPtr;
132     ibytes = strtoul(iBufBytes, &finalPtr, 10);
133     if (ibytes == 0) {
134         fprintf(stderr, "[Error] Incorrecta
135         cantidad de bytes para el buffer
136         de entrada.\n");
137         return ERROR_BYTES;
138     }
139 }
140
141 if (oBufBytes != NULL) {
142     char *finalPtr;
143     obytes = strtoul(oBufBytes, &finalPtr, 10);
144     if (obytes == 0) {
145         fprintf(stderr, "[Error] Incorrecta
146         cantidad de bytes para el buffer
147         de salida.\n");
148         return ERROR_BYTES;
149     }
150 }
151
152 if (pathInput == NULL || strcmp("-",pathInput) == 0)
153 {
154     inputFileDefault = TRUE;
155 }
156
157 if (pathOutput == NULL || strcmp("-",pathOutput) ==
158 0) {
159     outputFileDefault = TRUE;
160 }
161
162 if (inputFileDefault == FALSE) {

```

```

156         fileInput = fopen(pathInput, "r"); // Opens
           an existing text file for reading purpose
157
158         if (fileInput == NULL) {
           fprintf(stderr, "[Error] El archivo
           de input no pudo ser abierto para
           lectura: %s \n", pathInput);
159         return ERROR_FILE;
160     }
161 }
162
163 if (outputFileDefault == FALSE) {
164     fileOutput = fopen(pathOutput, "w"); //
           Opens a text file for writing. Pace the
           content.
165     if (fileOutput == NULL) {
166         fprintf(stderr, "[Error] El archivo
           de output no pudo ser abierto
           para escritura: %s \n",
           pathOutput);
167
168         if (inputFileDefault == FALSE) {
169             int result = fclose(
                 fileInput);
170             if (result == EOF) {
171                 fprintf(stderr, "[
                 Warning] El
                 archivo de input
                 no pudo ser
                 cerrado
                 correctamente: %s
                 \n", pathInput);
172             }
173         }
174
175         return ERROR_FILE;
176     }
177 }
178
179 int ifd = fileno(fileInput);
180 int ofd = fileno(fileOutput);
181
182 int executeResult = palindrome(ifd, ibytes, ofd,
           obytes);
183
184 int resultFileInputClose = 0; // EOF = -1
185
186 if (inputFileDefault == FALSE && fileInput != NULL)
187 {
188     resultFileInputClose = fclose(fileInput);
189     if (resultFileInputClose == EOF) {
           fprintf(stderr, "[Warning] El
           archivo de input no pudo ser
           cerrado correctamente: %s \n",

```

```

190         pathInput);
191     }
192 }
193     if (outputFileDefault == FALSE && fileOutput != NULL
194         ) {
195         int result = fclose(fileOutput);
196         if (result == EOF) {
197             fprintf(stderr, "[Warning] El
198                 archivo de output no pudo ser
199                 cerrado correctamente: %s \n",
200                 pathOutput);
201             resultFileInputClose = EOF;
202         }
203     }
204     if (resultFileInputClose != 0) {
205         return ERROR_FILE;
206     }
207     return executeResult;
208 }
209
210 int main(int argc, char **argv) {
211     // / -i lalala.txt -o pepe.txt -I 2 -O 3 => 9
212     // parameters as maximum
213     if (argc > 9) {
214         fprintf(stderr, "[Error] Cantidad máxima de
215             parámetros incorrecta: %d \n", argc);
216         return INCORRECT_QUANTITY_PARAMS;
217     }
218     return executeByMenu(argc, argv);
219 }

```

### 3.2. Código fuente en lenguaje C: process.c

```

1
2 #include "process.h"
3
4
5 #define FALSE 0
6 #define TRUE 1
7 #define LEXICO_BUFFER_SIZE 3
8
9 enum ParameterState {
10     OKEY = 0, INCORRECT_QUANTITY_PARAMS = 1, ERROR_MEMORY = 2, ERROR_READ = 3,
11     ERROR_WRITE = 4, LOAD_I_BUFFER = 5
12 };
13
14 size_t isize;
15 size_t osize;
16 int oFileDescriptor;
17 char * lexico = NULL;
18 int quantityCharacterInLexico = 0;
19 int savedInOFile = FALSE;
20 char * obuffer = NULL;
21 char * ibuffer = NULL;
22 int bytesLexico = 0;
23
24 char toLowerCase(char word) {
25     /* ASCII:
26     *
27         A - Z = [65 - 90]
28         a - z = [97 - 122]

```

```

27         *           0 - 9 = [48 - 57]
28         *           - = 45
29         *           - = 95
30         */
31         if (word >= 65 && word <= 90) {
32             word += 32;
33         }
34         return word;
35     }
36 }
37
38 int verifyPalindromic(char * word, int quantityCharacterInWord) {
39     if (word == NULL || quantityCharacterInWord <= 0) {
40         return FALSE;
41     }
42
43     if (quantityCharacterInWord == 1) {
44         // The word has one character
45         return TRUE;
46     }
47
48     if (quantityCharacterInWord == 2) {
49         char firstCharacter = toLowerCase(word[0]);
50         char lastCharacter = toLowerCase(word[1]);
51         if (firstCharacter != lastCharacter) {
52             return FALSE;
53         }
54     }
55     return TRUE;
56 }
57
58 double middle = (double)quantityCharacterInWord / 2;
59 int idx = 0;
60 int validPalindromic = TRUE;
61 int last = quantityCharacterInWord - 1;
62 while(idx < middle && last > middle && validPalindromic == TRUE) {
63     char firstCharacter = toLowerCase(word[idx]);
64     char lastCharacter = toLowerCase(word[last]);
65     if (firstCharacter != lastCharacter) {
66         validPalindromic = FALSE;
67     }
68
69     idx ++;
70     last --;
71 }
72
73 return validPalindromic;
74 }
75
76 int isKeywords(char character) {
77     /* ASCII:
78     *       A - Z = [65 - 90]
79     *       a - z = [97 - 122]
80     *       0 - 9 = [48 - 57]
81     *       - = 45
82     *       _ = 95
83     */
84     if ((character >= 65 && character <= 90) || (character >= 97 && character <=
85         122) || (character >= 48 && character <= 57) || character == 45 || character == 95) {
86         return TRUE;
87     }
88     return FALSE;
89 }
90
91 void * myRealloc(void * ptr, size_t tamanyoNew, int tamanyoOld) {
92     if (tamanyoNew <= 0) {
93         free(ptr);
94         ptr = NULL;
95         return NULL;
96     }
97
98     void * ptrNew = (void *) malloc(tamanyoNew);
99     if (ptrNew == NULL) {
100         return NULL;
101     }
102
103     if (ptr == NULL) {
104         return ptrNew;
105     }
106
107     int end = tamanyoNew;
108     if (tamanyoOld < tamanyoNew) {
109         end = tamanyoOld;
110     }
111
112     char *tmp = ptrNew;
113     const char *src = ptr;
114
115     while (end--) {
116         *tmp = *src;
117         tmp++;
118         src++;
119     }
120
121     free(ptr);

```



```

125         ptr = NULL;
126         return ptrNew;
127     }
128 }
129
130 void initializeBuffer(size_t bytes, char * buffer) {
131     // initialize the buffer
132     int i;
133     for(i = 0; i < bytes; ++i){
134         buffer[i] = '\0';
135     }
136 }
137
138 int writeBufferInOFile(int * amountSavedInBuffer, char * buffer) {
139     int completeDelivery = FALSE;
140     int bytesWriteAcum = 0;
141     int bytesToWrite = (*amountSavedInBuffer);
142     while (completeDelivery == FALSE) {
143         int bytesWrite = write(oFileDescriptor, buffer + bytesWriteAcum,
144                                bytesToWrite);
145         if (bytesWrite < 0) {
146             return ERROR_WRITE;
147         }
148         bytesWriteAcum += bytesWrite;
149         bytesToWrite = (*amountSavedInBuffer) - bytesWriteAcum;
150
151         if (bytesToWrite <= 0) {
152             completeDelivery = TRUE;
153         }
154     }
155     return OKEY;
156 }
157
158 int loadInLexico(char character) {
159     if (lexico == NULL) {
160         lexico = malloc(LEXICO_BUFFER_SIZE * sizeof(char));
161         bytesLexico = LEXICO_BUFFER_SIZE;
162     } else if (quantityCharacterInLexico >= bytesLexico) {
163         int bytesLexicoPreview = bytesLexico;
164         bytesLexico += LEXICO_BUFFER_SIZE;
165         lexico = myRealloc(lexico, bytesLexico * sizeof(char),
166                            bytesLexicoPreview);
167     }
168
169     if (lexico == NULL) {
170         fprintf(stderr, "[Error] Hubo un error en memoria (lexico). \n");
171         return ERROR_MEMORY;
172     }
173
174     lexico[quantityCharacterInLexico] = character;
175     quantityCharacterInLexico++;
176
177     return OKEY;
178 }
179
180 void copyFromLexicoToOBuffer(int * amountSavedInOBuffer) {
181     int i;
182     for (i = 0; i < quantityCharacterInLexico; ++i) {
183         obuffer[*amountSavedInOBuffer] = lexico[i];
184         *amountSavedInOBuffer = (*amountSavedInOBuffer) + 1;
185     }
186 }
187
188 char * loadBufferInitial(size_t size, char * buffer) {
189     buffer = (char *) malloc(size * sizeof(char));
190     if (buffer == NULL) {
191         fprintf(stderr, "[Error] Hubo un error de asignacion de memoria (
192             buffer). \n");
193         return NULL;
194     }
195
196     // initialize the buffer
197     initializeBuffer(size, buffer);
198
199     return buffer;
200 }
201
202 int processDataInIBuffer(char * ibuffer, int * amountSavedInOBuffer) {
203     int findEnd = FALSE;
204     int loadIBuffer = FALSE;
205     int idx = 0;
206     int rdo = OKEY;
207     while (findEnd == FALSE && loadIBuffer == FALSE) {
208         char character = ibuffer[idx];
209         if (character == '\0') {
210             findEnd = TRUE;
211         }
212
213         if (findEnd != TRUE && isKeywords(character) == TRUE) {
214             int rdo = loadInLexico(character);
215             if (rdo != OKEY) {
216                 return rdo;
217             }
218         } else if (quantityCharacterInLexico > 0) {
219             int itsPalindromic = verifyPalindromic(lexico,
220                                                     quantityCharacterInLexico);
221             if (itsPalindromic == TRUE) {

```

```

220         loadInLexico('\n');
221         int amountToSaved = (*amountSavedInOBuffer) +
222             quantityCharacterInLexico;
223         if ((*amountSavedInOBuffer) > 0 || savedInOFile ==
224             TRUE) {
225             amountToSaved++; // Es para el separador
226         }
227         if (amountToSaved > osize) {
228             /*
229              * Tomo la decision de pedir mas memoria
230              * para bajar el lexico completo
231              * y luego rearmo el buffer de salida y
232              * reinicio la cantidad guardada en 0.
233              */
234             obuffer = myRealloc(obuffer, amountToSaved*
235                 sizeof(char), (*amountSavedInOBuffer));
236             if (obuffer == NULL) {
237                 fprintf(stderr, "[Error] Hubo un
238                     error en memoria (obuffer). \n"
239                 );
240                 return ERROR_MEMORY;
241             }
242             copyFromLexicoToOBuffer(amountSavedInOBuffer
243             );
244             int rdoWrite = writeBufferInOFile(
245                 amountSavedInOBuffer, obuffer);
246             if (rdoWrite != OKEY) {
247                 return rdoWrite;
248             }
249             *amountSavedInOBuffer = 0;
250             savedInOFile = TRUE;
251             if (obuffer != NULL) {
252                 free(obuffer);
253                 obuffer = NULL;
254             }
255             obuffer = loadBufferInitial(osize, obuffer);
256             if (obuffer == NULL) {
257                 return ERROR_MEMORY;
258             }
259         } else {
260             copyFromLexicoToOBuffer(amountSavedInOBuffer
261             );
262         }
263     }
264     free(lexico);
265     lexico = NULL;
266     quantityCharacterInLexico = 0;
267 }
268 if ((idx + 1) == isize) {
269     loadIBuffer = TRUE;
270     rdo = LOAD_IBUFFER;
271 } else {
272     idx++;
273 }
274 return rdo;
275 }
276
277 enum IBufferState {
278     COMPLETE_DELIVERY = -1, END_I_FILE = -2, ERROR_I_READ = -3, OKEY_I_FILE =
279     -4
280 };
281
282 int loadIBufferWithIFile(size_t abytes, int ifd) {
283     int completeDelivery = FALSE;
284     int bytesReadAcum = 0;
285     int bytesToRead = abytes;
286     int end = FALSE;
287     // Lleno el buffer de entrada
288     while (completeDelivery == FALSE && end == FALSE) {
289         int bytesRead = read(ifd, ibuffer + bytesReadAcum, bytesToRead);
290         if (bytesRead == -1) {
291             fprintf(stderr, "[Error] Hubo un error en la lectura de
292                 datos del archivo. \n");
293             return ERROR_I_READ;
294         }
295         if (bytesRead == 0) {
296             end = TRUE;
297         }
298         bytesReadAcum += bytesRead;
299         bytesToRead = abytes - bytesReadAcum;
300         if (bytesToRead <= 0) {
301             completeDelivery = TRUE;
302         }
303     }
304     if (end == TRUE) {
305         return END_I_FILE;
306     }

```

```

307         return OKEY_ILFILE;
308     }
309 }
310
311 int cleanBuffers(int * amountSavedInOBuffer) {
312     int rdo = OKEY;
313     if (ibuffer != NULL) {
314         free(ibuffer);
315         ibuffer = NULL;
316     }
317
318     if (obuffer != NULL) {
319         if (amountSavedInOBuffer != NULL && (*amountSavedInOBuffer) > 0) {
320             int rdoWrite = writeBufferInOFile(amountSavedInOBuffer,
321                 obuffer);
322             if (rdoWrite != OKEY) {
323                 rdo = rdoWrite;
324             }
325         }
326         free(obuffer);
327         obuffer = NULL;
328     }
329
330     if (lexico != NULL) {
331         if (quantityCharacterInLexico > 0 && verifyPalindromic(lexico,
332             quantityCharacterInLexico) == TRUE) {
333             loadInLexico('\n');
334             int rdoWrite = writeBufferInOFile(&quantityCharacterInLexico,
335                 lexico);
336             if (rdoWrite != OKEY) {
337                 rdo = rdoWrite;
338             }
339             free(lexico);
340             lexico = NULL;
341         }
342     }
343     return rdo;
344 }
345
346 int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes) {
347     isize = ibytes;
348     osize = obytes;
349     oFileDescriptor = ofd;
350
351     ibuffer = loadBufferInitial(isize, ibuffer);
352     if (ibuffer == NULL) {
353         return ERROR_MEMORY;
354     }
355
356     obuffer = loadBufferInitial(osize, obuffer);
357     if (obuffer == NULL) {
358         free(ibuffer);
359         ibuffer = NULL;
360         return ERROR_MEMORY;
361     }
362
363     int * amountSavedInOBuffer = (int *) malloc(sizeof(int));
364     if (amountSavedInOBuffer == NULL) {
365         fprintf(stderr, "[Error] Hubo un error de asignacion de memoria (
366             amountSavedInOBuffer). \n");
367         free(ibuffer);
368         ibuffer = NULL;
369         free(obuffer);
370         obuffer = NULL;
371         return ERROR_MEMORY;
372     }
373     amountSavedInOBuffer[0] = 0;
374
375     int rdoProcess = OKEY;
376     int error = FALSE;
377     int rdoLoadIBuffer = OKEY_ILFILE;
378     while (rdoLoadIBuffer == OKEY_ILFILE && error == FALSE) {
379         rdoLoadIBuffer = loadIBufferWithIFile(ibytes, ifd);
380
381         if (ibuffer != NULL && ibuffer[0] != '\0') {
382             int resultProcessWrite = processDataInIBuffer(ibuffer,
383                 amountSavedInOBuffer);
384             if (resultProcessWrite == LOAD_ILBUFFER) {
385                 // initialize the ibuffer
386                 initializeBuffer(ibytes, ibuffer);
387             }
388             if (resultProcessWrite == ERROR_MEMORY || resultProcessWrite
389                 == ERROR_WRITE) {
390                 error = TRUE;
391                 rdoProcess = resultProcessWrite;
392             }
393         }
394     }
395
396     int rdoClean = cleanBuffers(amountSavedInOBuffer);
397
398     if (amountSavedInOBuffer != NULL) {
399         free(amountSavedInOBuffer);
400         amountSavedInOBuffer = NULL;
401     }

```

```

400         if (rdoClean != OKEY) {
401             return rdoClean;
402         }
403         return rdoProcess;
404     }
405 }

```

## 4. Código MIPS32

### 4.1. Código MIPS32: cleanBuffers.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define TRUE 1
7  #define DIR_NULL 0
8  #define LINE_BREAK 10
9
10 # Resultados de funciones posibles
11 #define OKEY 0
12
13
14 ##----- cleanBuffers -----##
15
16     .align 2
17     .globl cleanBuffers
18     .ent cleanBuffers
19 cleanBuffers:
20     .frame $fp,48,ra
21     .set noreorder
22     .cpload t9
23     .set reorder
24
25     #Stack frame creation
26     subu sp,sp,48
27     .cpstore 16
28     sw ra,40(sp)
29     sw $fp,36(sp)
30     sw gp,32(sp)
31     move $fp,sp
32
33     # Parameter
34     # Guardo en la direccion de memoria 48($fp)
35     # la variable * amountSavedInOBuffer
36     #(int * amountSavedInOBuffer).
37     sw a0,48($fp)
38
39     # Guardo en la direccion 24($fp) OKEY
40     #(=zero). Representa la variable rdo.
41     sw zero,24($fp)
42

```

```

43      lw v0,ibuffer
44      # If (ibuffer == NULL) goto FreeObuffer.
45      beq v0,DIR_NULL,$FreeObuffer
46
47
48      # Cargo en a0 ibuffer. Parametro de la funcion
        myfree.
49      lw a0,ibuffer
50
51      la t9,myfree
52      jal ra,t9      # Ejecuto la funcion myfree.
53      sw zero,ibuffer # ibuffer = NULL
54 $FreeObuffer:
55      lw v0,obuffer
56      # If (obuffer == NULL) goto FreeLexico.
57      beq v0,DIR_NULL,$FreeLexico
58      # (amountSavedInOBuffer != NULL &&
59      #      (*amountSavedInOBuffer) > 0) ?
60      # (amountSavedInOBuffer != NULL) ?
61
62      lw v0,48($fp) # Cargo en v0 amountSavedInOBuffer.
63
64      # If (amountSavedInOBuffer == NULL) goto
        FreeMyOBuffer
65      beq v0,DIR_NULL,$FreeMyOBuffer
66
67
68      # amountSavedInOBuffer is not NULL
69
70      # ((*amountSavedInOBuffer) > 0) ?
71      lw v0,48($fp) # Cargo en v0 amountSavedInOBuffer.
72      lw v0,0(v0)   # Cargo el contenido de lo apuntado
        por amountSavedInOBuffer en v0.
73      blez          v0,$FreeMyOBuffer
74      # If ((* amountSavedInOBuffer) <= 0) goto
        FreeMyOBuffer.
75
76      # (*amountSavedInOBuffer) is greater then 0
77
78      # int rdoWrite = writeBufferInOFile(
        amountSavedInOBuffer, obuffer);
79      lw          a0,48($fp)      # Cargo en a0
        amountSavedInOBuffer.Parametro de la funcion
        writeBufferInOFile.
80      lw          a1,obuffer      # Cargo en a1
        obuffer. Parametro de la funcion
        writeBufferInOFile.
81
82      la          t9,writeBufferInOFile
83      jal          ra,t9          # Ejecuto la funcion
        writeBufferInOFile.
84      sw          v0,28($fp)      # En v0 esta el resultado de
        writeOBufferInOFile (que seria la variable
        rdoWrite). Guardo esto en la direccion 28($fp).

```

```

85      # (rdoWrite != OKEY) ?
86      lw v0,28($fp) # Cargo en v0 rdoWrite.
87      beq v0,OKEY,$FreeMyOBuffer # If (rdoWrite ==
88          OKEY) goto FreeMyOBuffer.
89
90      # rdoWrite is not OKEY.
91
92      # rdo = rdoWrite;
93      lw v0,28($fp) # Cargo en v0 rdoWrite.
94      sw v0,24($fp) # Asigno a la variable rdo rdoWrite.
95 $FreeMyOBuffer:
96      lw a0,obuffer # Cargo en a0 obuffer. Parametro de
97          la funcion myfree.
98      la t9,myfree
99      jal ra,t9 # Ejecuto la funcion myfree.
100     sw zero,obuffer # obuffer = NULL
101 $FreeLexico:
102     # (lexico != NULL) ?
103     lw v0,lexico
104     beq v0,DIR_NULL,$ReturnCleanBuffers # If (lexico
105         == NULL) goto ReturnCleanBuffers.
106
107     # lexico is not NULL
108     lw v0,quantityCharacterInLexico
109     blez v0,$FreeMyLexico # If (
110         quantityCharacterInLexico <= 0) goto FreeMyLexico
111
112     lw a0,lexico
113     lw a1,quantityCharacterInLexico
114     la t9,verifyPalindromic
115     jal ra,t9 # Ejecuto verifyPalindromic.
116     Verifico si lo que quedo en lexico es palindromo.
117
118     move v1,v0
119     li v0,TRUE
120     bne v1,v0,$FreeMyLexico # If no es
121         palindromo, goto FreeMyLexico
122
123     li a0,LINE_BREAK
124     la t9,loadInLexico
125     jal ra,t9 # Ejecuto loadInLexico. Agrego el
126         salto de linea al lexico.
127
128     # int rdoWrite = writeBufferInOFile(&
129         quantityCharacterInLexico, lexico);
130     la a0,quantityCharacterInLexico
131     lw a1,lexico
132     la t9,writeBufferInOFile
133     jal ra,t9 # Ejecuto writeBufferInOFile
134         para guardar el lexico que es palindromo.
135     sw v0,28($fp) # Guardo en la direccion 28(
136         $fp) el resultado de la funcion
137         writeBufferInOFile.

```

```

128      lw      v0,28($fp)
129
130      beq      v0,OKEY,$FreeMyLexico # Si el resultado de
      escribir en el archivo de salida no da error,
      salto a FreeMyLexico.
131
132      lw      v0,28($fp) # Cargo en v0 el
      error de escribir en el file de salida.
133      sw      v0,24($fp) # Guardo este
      codigo de error en la direccion 24($fp).
134 $FreeMyLexico:
135      lw      a0,lexico
136      la      t9,myfree
137      jal     ra,t9 # Ejecuto myfree para lexico.
138      sw      zero,lexico # lexico = NULL.
139 $ReturnCleanBuffers:
140      lw      v0,24($fp)
141      move     sp,$fp
142      lw      ra,40(sp)
143      lw      $fp,36(sp)
144      addu     sp,sp,48
145      j      ra
146      .end     cleanBuffers

```

Stack frame:

Offset	Contents	Type reserved area	Comment
48	*amountSavedInOBuffer		
44			nothing to keep
40	ra	SRA	
36	fp		
32	gp		
28	rdoWrite		Resultado de la función writeBufferInOFile: OKEY   Error
24	Resultado de la función	LTA	OKEY    rdoWrite
20			nothing to keep
16			nothing to keep
12	a3	ABA	Invocación a myfree: 1) buffer -> a0 2) obuffer -> a0 3) lexico -> a0
8	a2		Invocación a writeBufferInOFile: 1) * amountSavedInOBuffer -> a0    obuffer -> a1 2) quantityCharacterInLexico -> a0    lexico -> a1
4	a1		Invocación a verifyPalindromic: 1) lexico -> a0    quantityCharacterInLexico -> a1
0	a0		Inicialmente contiene el valor del parametro *amountSavedInOBuffer. Invocación a loadInLexico: 1) '\n' -> a0

Figura 1: Stack frame: cleanBuffers

## 4.2. Código MIPS32: copyFromLexicoToOBuffer.S

```

1 #include <mips/regdef.h>
2 #include <sys/syscall.h>
3
4 #STATICS VAR DEFINITIONS
5
6 #define FALSE 0
7 #define TRUE 1
8
9

```

```

10  ##----- copyFromLexicoToOBuffer -----##
11
12      .align          2
13      .globl          copyFromLexicoToOBuffer
14      .ent            copyFromLexicoToOBuffer
15  copyFromLexicoToOBuffer:
16      .frame          $fp,24,ra
17      .set            noreorder
18      .cpload        t9
19      .set            reorder
20
21      #Stack frame creation
22      subu            sp,sp,24
23
24      .cprestore 0
25      sw              $fp,20(sp)
26      sw              gp,16(sp)
27      move            $fp,sp
28
29      # Parameter
30      sw              a0,24($fp) # Guardo en la direccion
                                de memoria 24($fp) la variable *
                                amountSavedInOBuffer (int * amountSavedInOBuffer)
31      .
32      sw              zero,8($fp) # Guardo en la
                                direccion 8($fp) el contador i para el for
                                inicializado en 0.
33  $ForCopy:
34      lw              v0,8($fp) # Cargo en v0 el contador
                                i.
35      lw              v1,quantityCharacterInLexico
36      slt             v0,v0,v1 # Guardo TRUE en v0 si (i
                                < quantityCharacterInLexico), sino guardo FALSE.
37      bne             v0,FALSE,$InForCopy # Si el
                                resultado de la comparacion no es FALSE, o sea, (
                                i < quantityCharacterInLexico), entro al for (
                                goto InForCopy).
38      b               $ReturnCopyFromLexicoToOBuffer #
                                Salto incondicional a
                                ReturnCopyFromLexicoToOBuffer (el return de la
                                funcion).
39  $InForCopy:
40      # obuffer[*amountSavedInOBuffer] = lexico[i];
41      lw              v0,24($fp) # Cargo *
                                amountSavedInOBuffer en v0.
42      lw              v1,obuffer
43      lw              v0,0(v0) # Cargo en v0 lo
                                almacenado en la direccion de memoria guardada en
                                v0 (*amountSavedInOBuffer).
44      addu            a0,v1,v0 # Guardo en a0 la nueva
                                direccion de memoria sobre obuffer:
45      # obuffer + *amountSavedInOBuffer = obuffer[*
                                amountSavedInOBuffer]

```



```

46      lw          v1,lexico # Cargo en v1 lexico.
47      lw          v0,8($fp) # Cargo en v0 el indice i
      guardado en la direccion de memoria 8($fp).
48      addu        v0,v1,v0 # Me muevo sobre lexico:
      lexico + i = lexico[i]. Guardo la direccion en v0
      .
49      lbu          v0,0(v0) # Cargo en v0 lo guardado
      en la direccion de memoria almacenada en v0 (es
      sobre lexico).
50      sb          v0,0(a0) # Guardo en la direccion
      de memoria almacenada en a0 (es sobre obuffer) lo
      almacenado en v0. 0 sea:
51      # obuffer[*amountSavedIn0Buffer] = lexico[i];
52
53      # *amountSavedIn0Buffer = (*amountSavedIn0Buffer) +
      1;
54      lw          v1,24($fp) # Cargo *
      amountSavedIn0Buffer en v1.
55      lw          v0,24($fp) # Cargo *
      amountSavedIn0Buffer en v0.
56      lw          v0,0(v0) # Cargo en v0 lo
      almacenado en la direccion de memoria guardada en
      v0 (*amountSavedIn0Buffer).
57      addu        v0,v0,1 # Incremento en 1.
58      sw          v0,0(v1) # Guardo el nuevo valor de
      amountSavedIn0Buffer.
59
60      # ++i
61      lw          v0,8($fp) # Cargo en v0 el indice i
      guardado en la direccion de memoria 8($fp).
62      addu        v0,v0,1 # Incremento en 1 el
      indice i.
63      sw          v0,8($fp) # Guardo el incremento.
64
65      b            $ForCopy # Salto incondicional.
      Vuelvo al comienzo del loop for.
66      $ReturnCopyFromLexicoTo0Buffer:
67      move        sp,$fp
68      lw          $fp,20(sp)
69      addu        sp,sp,24
70      j            ra
71      .end        copyFromLexicoTo0Buffer

```

Stack frame:

void copyFromLexicoToOBuffer(int * amountSavedInOBuffer)			
Offset	Contents	Type reserved area	Comment
24	ra    * amountSavedInOBuffer	SRA	
20	fp		
16	gp		
12	a3	ABA	
8	a2    i		
4	a1		
0	a0		Inicialmente contiene el valor del parametro * amountSavedInOBuffer.

Figura 2: Stack frame: copyFromLexicoToOBuffer

### 4.3. Código MIPS32: initializeBuffer.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FALSE          0
7  #define TRUE           1
8
9
10 ##----- initializeBuffer -----##
11
12     .align             2
13     .globl             initializeBuffer
14     .ent               initializeBuffer
15 initializeBuffer:
16     .frame             $fp,24,ra
17     .set               noreorder
18     .cpload            t9
19     .set               reorder
20
21     #Stack frame creation
22     subu               sp,sp,24
23
24     .cprestore 0
25     sw                 $fp,20(sp)
26     sw                 gp,16(sp)
27     move               $fp,sp
28
29     # Parameters
30     sw                 a0,24($fp) # Guardo en la direccion
                                de memoria 24($fp) la variable bytes (size_t
                                bytes).
31     sw                 a1,28($fp) # Guardo en la direccion
                                de memoria 28($fp) la variable buffer (char *
                                buffer).
32
33     sw                 zero,8($fp) # Guardo en la
                                direccion de memoria 8($fp) el contenido 0, que
                                seria la variable i (int i;).

```

```

34 $ForInitializeBuffer:
35     lw          v0,8($fp) # Cargo en v0 el
                               contenido en la direccion de memoria 8($fp), que
                               seria la variable i.
36     lw          v1,24($fp) # Cargo en v1 el
                               contenido en la direccion de memoria 24($fp), que
                               seria la variable bytes.
37     sltu        v0,v0,v1 # Comparo i (v0) con bytes
                               (v1). Si i < bytes, guardo TRUE en v0, sino
                               guardo FALSE.
38     bne         v0,FALSE,$ForInitializeCharacter
                               # If (i < bytes) goto
                               ForInitializeCharacter.
39     b           $InitializeBufferReturn
                               # Salto incondicional al return de la
                               funcion initializeBuffer.
40 $ForInitializeCharacter:
41     # buffer[i] = '\0';
42     lw          v1,28($fp) # Cargo en v1 el
                               contenido en la direccion de memoria 28($fp), que
                               seria la variable * buffer.
43     lw          v0,8($fp) # Cargo en v0 el
                               contenido en la direccion de memoria 8($fp), que
                               seria la variable i.
44     addu        v0,v1,v0 # Me corro en el buffer la
                               cantidad estipulada por la variable i (buffer[i]
                               = buffer + i), y lo guardo en v0.
45     sb          zero,0(v0) # Guardo '\0' = 0 en la
                               posicion del buffer estipulada previamente (
                               buffer[i] = '\0';).
46
47     # ++ i
48     lw          v0,8($fp) # Cargo en v0 el
                               contenido en la direccion de memoria 8($fp), que
                               seria la variable i.
49     addu        v0,v0,1      # Incremento en 1 la
                               variable i (i ++).
50     sw          v0,8($fp) # Guardo en la direccion
                               de memoria 8($fp) el nuevo valor de la variable i
                               .
51
52     b           $ForInitializeBuffer
                               # Vuelvo a entrar en el for (bucle).
53 $InitializeBufferReturn:
54     move        sp,$fp
55     lw          $fp,20(sp)
56     addu        sp,sp,24
57     j           ra          # Jump and return
58     .end        initializeBuffer

```

Stack frame:

void initializeBuffer(size_t bytes, char * buffer)			
Offset	Contents	Type reserved area	Comment
28	* buffer	SRA	
24	ra    bytes		
20	fp		
16	gp		
12	a3	ABA	
8	a2    i		
4	a1		Inicialmente contiene el valor del parametro * buffer.
0	a0		Inicialmente contiene el valor del parametro bytes.

Figura 3: Stack frame: initializeBuffer

#### 4.4. Código MIPS32: isKeywords.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FALSE          0
7  #define TRUE           1
8
9
10 ##----- isKeywords -----##
11
12     .align             2
13     .globl             isKeywords
14     .ent               isKeywords
15 isKeywords:
16     .frame             $fp,24,ra
17     .set               noreorder
18     .cpload            t9
19     .set               reorder
20
21     #Stack frame creation
22     subu               sp,sp,24
23     .cpstore 0
24     sw                 $fp,20(sp)
25     sw                 gp,16(sp)
26     move               $fp,sp
27
28     move               v0,a0    # Muevo de a0 a v0 el
                                parametro de la funcion (char character).
29     sb                 v0,8($fp) # Guardo en la direccion
                                de memoria 8($fp) el contenido de la variable
                                character que se encuentra en el registro v0.
30     lb                 v0,8($fp) # Cargo el byte
                                character en v0 que estaba en la direccion de
                                memoria 8($fp).
31
32     # character >= 65 && character <= 90    ---    A - Z
                                = [65 - 90]

```

```

33      slt          v0,v0,65      # Compara el contenido de
      la variable character con el literal 65, y
      guarda true en v0 si
34          # el primero (character) es mas
      chico que el segundo (65).
35      bne          v0,FALSE,$VerifyCharacterOfaToz
      # Si no es igual a FALSE, o sea,
      character < 65, salta a VerifyCharacterOfaToz.
36      lb          v0,8($fp)      # Cargo el byte
      character en v0 que estaba en la direccion de
      memoria 8($fp).
37      slt          v0,v0,91      # Compara el contenido de
      la variable character con el literal 91, y
      guarda true en v0 si el
38          # primero (character) es mas chico
      que el segundo (91).
39      bne          v0,FALSE,$ReturnIsKeywordsTrue
      # Si no es igual a FALSE, o sea,
      character > 91, salta a ReturnIsKeywordsTrue.
40 $VerifyCharacterOfaToz:
41      lb          v0,8($fp)      # Cargo el byte
      character en v0 que estaba en la direccion de
      memoria 8($fp).
42
43      # character >= 97 && character <= 122      ---      a - z
      = [97 - 122]
44      slt          v0,v0,97      # Compara el contenido de
      la variable character con el literal 97, y
      guarda true en v0 si
45          # el primero (character) es mas
      chico que el segundo (97).
46      bne          v0,FALSE,$VerifyCharacterOf0To9
      # Si no es igual a FALSE, o sea,
      character < 65, salta a VerifyCharacterOf0To9.
47      lb          v0,8($fp)      # Cargo el byte
      character en v0 que estaba en la direccion de
      memoria 8($fp).
48      slt          v0,v0,123     # Compara el contenido
      de la variable character con el literal 123, y
      guarda true en v0 si el
49          # primero (character) es mas chico
      que el segundo (123).
50      bne          v0,FALSE,$ReturnIsKeywordsTrue
      # Si no es igual a FALSE, o sea,
      character > 123, salta a ReturnIsKeywordsTrue.
51 $VerifyCharacterOf0To9:
52      lb          v0,8($fp)      # Cargo el byte
      character en v0 que estaba en la direccion de
      memoria 8($fp).
53
54      # character >= 48 && character <= 57      ---      0 - 9
      = [48 - 57]
55      slt          v0,v0,48      # Compara el contenido de
      la variable character con el literal 48, y

```

```

56         guarda true en v0 si el
           # primero (character) es mas chico
           que el segundo (48).
57     bne          v0,zero,$VerifyCharacterGuionMedio
           # Si no es igual a FALSE, o sea, character
           < 48, salta a VerifyCharacterGuionMedio.
58     lb          v0,8($fp)    # Cargo el byte
           character en v0 que estaba en la direccion de
           memoria 8($fp).
59     slt         v0,v0,58     # Compara el contenido de
           la variable character con el literal 58, y
           guarda true en v0 si el
           # primero (character) es mas chico
           que el segundo (58).
60     bne          v0,zero,$ReturnIsKeywordsTrue
           # Si no es igual a FALSE, o sea,
           character > 58, salta a ReturnIsKeywordsTrue.
61 $VerifyCharacterGuionMedio:
62     lb          v1,8($fp)    # Cargo el byte
           character en v1 que estaba en la direccion de
           memoria 8($fp).
63
64
65     # character == 45    ---    - = 45
66     li          v0,45        # Cargo el literal 45 en v0
           para hacer luego la comparacion.
67     beq         v1,v0,$ReturnIsKeywordsTrue
           # If (character == 45) goto
           ReturnIsKeywordsTrue
68
69     lb          v1,8($fp)    # Cargo el byte
           character en v1 que estaba en la direccion de
           memoria 8($fp).
70
71     # character == 95    ---    _ = 95
72     li          v0,95        # Cargo el literal 95 en v0
           para hacer luego la comparacion.
73     beq         v1,v0,$ReturnIsKeywordsTrue
           # If (character == 95) goto
           ReturnIsKeywordsTrue
74     b           $ReturnIsKeywordsFalse
           # Salto incondicional para retornar FALSE
           (character no es un keyword).
75 $ReturnIsKeywordsTrue:
76     li          v0,TRUE      # Cargo en v0 TRUE (que
           seria igual a 1).
77     sw          v0,12($fp)   # Guardo el resultado
           de la funcion TRUE (v0) en la direccion de
           memoria 12($fp).
78     b           $ReturnIsKeywords
           # Salto incondicional para retornar
           resultado de las comparaciones.
79 $ReturnIsKeywordsFalse:
80     sw          zero,12($fp)  # Guardo FALSE (que
           seria igual a 0) en la direccion de memoria 12(

```

```

81      $fp).
$ReturnIsKeywords:
82      lw          v0,12($fp)    # Cargo en v0 el
                                resultado de la funcion isKeywords guardado en la
                                direccion de memoria 12($fp).
83      move        sp,$fp
84      lw          $fp,20(sp)
85      addu        sp,sp,24
86      j          ra    # Jump and return
87      .end        isKeywords

```

Stack frame:

int isKeywords(char character)			
Offset	Contents	Type reserved area	Comment
24	ra	SRA	
20	fp		
16	gp		
12	a3    Resultado de la función	ABA	Resultado de la función: TRUE    FALSE
8	a2    character		
4	a1		
0	a0		Inicialmente contiene el valor del parametro character.

Figura 4: Stack frame: isKeywords

## 4.5. Código MIPS32: loadBufferInitial.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FILE_DESCRIPTOR_STDERR  2
7
8
9  # Size mensaje
10 #define BYTES_MENSAJE_ERROR_MEMORIA_BUFFER  59
11
12
13 ##----- loadBufferInitial -----##
14
15      .align          2
16      .globl          loadBufferInitial
17      .ent             loadBufferInitial
18 loadBufferInitial:
19      .frame          $fp,48,ra
20      .set             noreorder
21      .cpload         t9
22      .set             reorder
23
24      #Stack frame creation
25      subu             sp,sp,48
26      .cprestore 16
27      sw              ra,40(sp)

```

```

28      sw          $fp,36(sp)
29      sw          gp,32(sp)
30      move        $fp,sp
31
32      # Parameters
33      sw          a0,48($fp) # Guardo en la direccion
                        de memoria 48($fp) la variable size (size_t size
34      sw          a1,52($fp) # Guardo en la direccion
                        de memoria 52($fp) la variable * buffer (char *
                        buffer).
35
36      lw          a0,48($fp) # Cargo en a0 la
                        variable size, parametro de mymalloc.
37      la          t9,mymalloc
38      jal         ra,t9     # Ejecuto mymalloc.
39      sw          v0,52($fp) # Asigno la memoria
                        reservada con mymalloc a buffer.
40
41      lw          v0,52($fp)
42      bne         v0,zero,$InitializeMemory
                        # If (buffer != NULL) goto
                        InitializeMemory.
43
44      # buffer is NULL => Mensaje de error
45      li          a0,FILE_DESCRIPTOR_STDERR
                        # Cargo en a0 FILE_DESCRIPTOR_STDERR.
46      la          a1,MENSAJE_ERROR_MEMORIA_BUFFER
                        # Cargo en a1 la direccion de memoria
                        donde se encuentra el mensaje a cargar.
47      li          a2,
                        BYTES_MENSAJE_ERROR_MEMORIA_BUFFER # Cargo en
                        a2 la cantidad de bytes a escribir.
48      li          v0, SYS_write
49      syscall     # No controlo error porque
                        sale de por si de la funcion con null y se
                        controla error luego.
50
51      sw          zero,24($fp) # Guardo NULL en la
                        direccion de memoria 24($fp).
52      b           $ReturnLoadBufferInitial
                        # Salto al return de la funcion.
53 $InitializeMemory:
54      lw          a0,48($fp) # Cargo en a0 la
                        variable size. Parametro de la funcion
                        initializeBuffer.
55      lw          a1,52($fp) # Cargo en a0 la
                        variable buffer. Parametro de la funcion
                        initializeBuffer.
56      la          t9,initializeBuffer
57      jal         ra,t9     # Ejecuto la funcion
                        initializeBuffer.
58      lw          v0,52($fp) # Cargo en v0 la
                        variable buffer.

```



```

59         sw            v0,24($fp) # Guardo buffer en la
           direccion de memoria 24($fp).
60 $ReturnLoadBufferInitial:
61         lw            v0,24($fp)
62         move          sp,$fp
63         lw            ra,40(sp)
64         lw            $fp,36(sp)
65         addu          sp,sp,48
66         j             ra
67         .end          loadBufferInitial
68
69
70 ## Mensajes de error
71
72         .rdata
73
74         .align 2
75 MENSAJE_ERROR_MEMORIA_BUFFER:
76         .ascii "[Error] Hubo un error de asignacion de
           memoria (buffer)"
77         .ascii ". \n\000"

```

Stack frame:

char * loadBufferInitial(size_t size, char * buffer)				
Offset	Contents	Type reserved area	Comment	
52	* buffer			
48	size			
44			nothing to keep	
40	ra	SRA		
36	fp			
32	gp			
28				
24	Resultado de la función	LTA	NULL o puntero a buffer	
20			nothing to keep	
16			nothing to keep	
12	a3	ABA	Cada vez que se invoca a SYS_write (para informar errores), se guarda en a3 si hubo o no error.	Invocación a mymalloc: 1) size -> a0
8	a2			
4	a1		Inicialmente contiene el valor del parametro * buffer.	
0	a0		Inicialmente contiene el valor del parametro size.	

Figura 5: Stack frame: loadBufferInitial

## 4.6. Código MIPS32: loadIBufferWithIFile.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FALSE 0
7  #define TRUE 1
8  #define FILE_DESCRIPTOR_STDERR 2
9
10 # Resultados de funciones posibles
11 #define COMPLETE_DELIVERY -1
12 #define END_I_FILE -2

```

```

13 #define ERROR_I_READ -3
14 #define OKEY_I_FILE -4
15
16 # Size mensajes
17 #define BYTES_MENSAJE_ERROR_Lectura_ARCHIVO 60
18
19
20 ##----- loadIBufferWithIFile -----##
21
22 .align 2
23 .globl loadIBufferWithIFile
24 .ent loadIBufferWithIFile
25 loadIBufferWithIFile:
26 .frame $fp,64,ra
27 .set noreorder
28 .cload t9
29 .set reorder
30
31 #Stack frame creation
32 subu sp,sp,64
33
34 .cprestore 16
35 sw ra,56(sp)
36 sw $fp,52(sp)
37 sw gp,48(sp)
38 move $fp,sp
39
40 # Parameters
41 sw a0,64($fp) # Guardo en la direccion
de memoria 64($fp) la variable abytes (size_t
abytes).
42 sw a1,68($fp) # Guardo en la direccion
de memoria 68($fp) la variable ifd (int ifd).
43
44 sw zero,24($fp) # Guardo en la
direccion 24($fp) cero, que representa a la
variable completeDelivery.
45 sw zero,28($fp) # Guardo en la
direccion 28($fp) cero, que representa a la
variable bytesReadAcum.
46 lw v0,64($fp)
47 sw v0,32($fp) # Guardo en la direccion
32($fp) abytes, que representa a la variable
bytesToRead.
48 sw zero,36($fp) # Guardo en la
direccion 36($fp) cero, que representa a la
variable end.
49
50 # Lleno el buffer de entrada
51 $WhileLoadIBuffer:
52 # (completeDelivery == FALSE) ?
53 lw v0,24($fp) # Cargo en v0
completeDelivery.

```

```

54      bne          v0,FALSE,
        $VerifyResultWhileLoadIBuffer # If (
        completeDelivery != FALSE) goto
        VerifyResultWhileLoadIBuffer.
55
56      # completeDelivery is FALSE
57
58      # (end == FALSE) ?
59      lw          v0,36($fp) # Cargo en v0 end.
60      bne          v0,FALSE,
        $VerifyResultWhileLoadIBuffer # If (
        completeDelivery != FALSE) goto
        VerifyResultWhileLoadIBuffer.
61
62      # Entre al while
63
64      # int bytesRead = read(ifd, ibuffer + bytesReadAcum,
        bytesToRead);
65      lw          v1,ibuffer
66      lw          v0,28($fp) # Cargo en v0
        bytesReadAcum.
67      addu        v0,v1,v0 # Guardo en v0 la
        direccion resultante de ibuffer+bytesReadAcum.
68      lw          a0,68($fp) # Cargo en v0 ifd.
69      move        a1,v0 # Cargo en a1 ifd. Parametro
        de la funcion read.
70      lw          a2,32($fp) # Cargo en a2
        bytesToRead. Parametro de la funcion read.
71
72      li          v0, SYS_read
73      syscall      # Seria read: int bytesRead
        = read(ifd, ibuffer + bytesReadAcum, bytesToRead)
        ;
74
75      # Controllo errores y cantidad de bytes leidos. v0
        contiene el numero de caracteres leidos (es
        negativo si hubo error y es 0 si llego a fin del
        archivo).
76
77      beq          a3,zero,$SavedBytesRead
        #Si a3 es cero, no hubo error
78
79      sw          v0,40($fp) # Guardo en la direccion
        de memoria 40($fd) el resultado de la funcion
        read, que
80      # estaria representado por la variable bytesRead.
81
82      # bytesRead == -1 ?
83      lw          v1,40($fp) # Cargo en v1 bytesRead.
84      li          v0,-1 # Cargo en v0 -1 para la
        comparacion.
85      bne          v1,v0,$ContinueValidationResultRead
        # If (bytesRead != -1) goto
        ContinueValidationResultRead.

```

```

86
87     # bytesRead is -1 => Mensaje de error.
88     li          a0,FILE_DESCRIPTOR_STDERR
89     # Cargo en a0 FILE_DESCRIPTOR_STDERR.
90     la          a1,MENSAJE_ERROR_LECTURA_ARCHIVO
91     # Cargo en a1 la direccion de memoria
92     # donde se encuentra el mensaje a cargar.
93     li          a2,
94     BYTES_MENSAJE_ERROR_LECTURA_ARCHIVO # Cargo en
95     a2 la cantidad de bytes a escribir.
96     li          v0, SYS_write
97     syscall      # No controlo error porque
98     # sale de por si de la funcion por error.
99
100    li          v0,ERROR_I_READ
101    sw          v0,44($fp) #
102    # Guardo el codigo de error en la direccion 44($fp)
103    .
104    b          $ReturnLoadIBufferWithIFile #
105    # Salto incondicional al return de la funcion.
106    $SavedBytesRead:
107    sw          v0,40($fp) # Guardo en la direccion
108    # de memoria 40($fd) el resultado de la funcion
109    # read, que
110    # estaria representado por la variable bytesRead.
111    $ContinueValidationResultRead:
112    lw          v0,40($fp) #
113    # Cargo en v1 bytesRead.
114    bne         v0,zero,
115    $ContinueAccumulatingBytesRead # If (bytesRead
116    # != 0) goto ContinueAccumulatingBytesRead
117    li          v0,TRUE
118    sw          v0,36($fp) # Asigno a la variable
119    # end, guardada en 36($fp), TRUE.
120    $ContinueAccumulatingBytesRead:
121    # bytesReadAcum += bytesRead;
122    lw          v1,28($fp) # Cargo en v1
123    # bytesReadAcum.
124    lw          v0,40($fp) # Cargo en v0 bytesRead.
125    addu        v0,v1,v0 # Sumo bytesReadAcum con
126    # bytesRead y guardo resultado en v0.
127    sw          v0,28($fp) # Guardo el resultado de
128    # la suma en bytesReadAcum.
129
130    # bytesRead = ibytes - bytesReadAcum;
131    lw          v1,64($fp) # Cargo en v1 ibytes.
132    lw          v0,28($fp) # Cargo en v0
133    # bytesReadAcum.
134    subu        v0,v1,v0 # Resto ibytes con
135    # bytesReadAcum y guardo resultado en v0, para
136    # saber cuandos bytes restan por leer del archivo.
137    sw          v0,32($fp) # Asigno a bytesRead
138    # el resultado de la resta.

```

```

118     # bytesToRead == 0 ?
119     lw          v0,32($fp)  # Cargo en v0
120     bytesToRead.
121     bne         v0,zero,$WhileLoadIBuffer
122     # If (bytesToRead != 0) goto
123     WhileLoadIBuffer
124
125     # bytesToRead is 0.
126     li          v0,TRUE
127     sw          v0,24($fp)  # Asigno a
128     completeDelivery TRUE.
129     b           $WhileLoadIBuffer
130     # Salto incondicional al comienzo del
131     while para cargar el buffer con los datos
132     # del archivo (goto WhileLoadIBuffer).
133 $VerifyResultWhileLoadIBuffer:
134     lw          v1,36($fp)  # Cargo en v1 end.
135     li          v0,TRUE
136     bne         v1,v0,$ReturnFunctionOkey
137     # If (end != TRUE) goto
138     ReturnFunctionOkey
139
140     li          v0,END_I_FILE
141     sw          v0,44($fp)  # Guardo codigo de error
142     en la direccion de memoria 44($fp).
143     b           $ReturnLoadIBufferWithIFile
144 $ReturnFunctionOkey:
145     li          v0,OKEY_I_FILE
146     sw          v0,44($fp)  # Guardo codigo de error
147     en la direccion de memoria 44($fp).
148 $ReturnLoadIBufferWithIFile:
149     lw          v0,44($fp)
150     move        sp,$fp
151     lw          ra,56(sp)
152     lw          $fp,52(sp)
153     addu        sp,sp,64
154     j           ra
155     .end        loadIBufferWithIFile
156
157 ## Mensajes de error
158
159     .rdata
160
161     .align 2
162 MENSAJE_ERROR_LECTURA_ARCHIVO:
163     .ascii "[Error] Hubo un error en la lectura de
164     datos del archivo"
165     .ascii ". \n\000"

```

Stack frame:

int loadBufferWithFile(size_t abytes, int ifd)			
Offset	Contents	Type reserved area	Comment
68	ifd		
64	abytes		
60			nothing to keep
56	ra	SRA	
52	fp		
48	gp		
44	Resultado de la función		ERROR_READ    OKEY_FILE    END_FILE
40	bytesRead	LTA	Resultado de la función SYS_read (variable bytesRead)
36	end		FALSE    TRUE
32	bytesToRead		Inicialmente igual a abytes
28	bytesReadAcum		Inicialmente en 0
24	completeDelivery		FALSE    TRUE
20			nothing to keep
16			nothing to keep
12	a3	ABA	Contiene si hubo error o no cuando se invocó a SYS_read y SYS_write (se usa para guardar mensaje de error).
8	a2		
4	a1		Inicialmente contiene el valor del parametro ifd.
0	a0		Inicialmente contiene el valor del parametro abytes.

Figura 6: Stack frame: loadBufferWithIFile

## 4.7. Código MIPS32: loadInLexico.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FALSE 0
7  #define TRUE 1
8  #define FILE_DESCRIPTOR_STDERR 2
9  #define LEXICO_BUFFER_SIZE 10
10
11 # Resultados de funciones posibles
12 #define ERROR_MEMORY 2
13
14 # Size mensajes
15 #define BYTES_MENSAJE_ERROR_MEMORIA_LEXICO 45
16
17
18
19 ##----- loadInLexico -----##
20
21     .align 2
22     .globl loadInLexico
23     .ent loadInLexico
24 loadInLexico:
25     .frame $fp,56,ra
26     .set noreorder
27     .cload t9
28     .set reorder
29
30     #Stack frame creation

```

```

31      subu          sp,sp,56
32
33      .cprestore 16
34      sw            ra,48(sp)
35      sw            $fp,44(sp)
36      sw            gp,40(sp)
37      move          $fp,sp
38      move          v0,a0
39
40      # Parameter
41      sb            v0,24($fp) # Guardo en la direccion
                        de memoria 24($fp) la variable character (char
                        character).
42      lw            v0,lexico # Cargo en v0 la variable
                        lexico.
43      bne           v0,zero,$VerifyIfReallocLexico
44      li            a0,LEXICO_BUFFER_SIZE
45      la            t9,mymalloc
46      jal           ra,t9
47      sw            v0,lexico
48      li            v0,LEXICO_BUFFER_SIZE
49      sw            v0,bytesLexico # Guardo
                        LEXICO_BUFFER_SIZE en la variable bytesLexico.
50      b             $VerifyIfCanLoadInLexico
51 $VerifyIfReallocLexico:
52      # (quantityCharacterInLexico >= bytesLexico) ?
53      lw            v0,quantityCharacterInLexico
54      lw            v1,bytesLexico
55      slt           v0,v0,v1 # Guarda en v0 TRUE si
                        quantityCharacterInLexico < bytesLexico, sino
                        guarda FALSE
56      bne           v0,zero,$VerifyIfCanLoadInLexico
                        # If (quantityCharacterInLexico <
                        bytesLexico) goto VerifyIfCanLoadInLexico.
57      # Se va por el negativo para no tener que alocar memoria
                        para guardar TRUE.
58
59      # quantityCharacterInLexico >= bytesLexico
60
61      # bytesLexico += LEXICO_BUFFER_SIZE;
62      lw            v0,bytesLexico
63      sw            v0,28($fp)
64      lw            v0,bytesLexico
65      addu          v0,v0,LEXICO_BUFFER_SIZE
66      sw            v0,bytesLexico
67
68      # lexico = myRealloc(lexico, bytesLexico*sizeof(char
                        ), bytesLexicoPreview);
69      lw            a0,lexico
70      lw            a1,bytesLexico
71      lw            a2,28($fp)
72      la            t9,myRealloc
73      jal           ra,t9
74      sw            v0,lexico

```

```

75 $VerifyIfCanLoadInLexico:
76     lw          v0,lexico
77     bne         v0,zero,$LoadCharacterInLexico
           # If (lexico != NULL) goto
           LoadCharacterInLexico.
78
79     # Hubo problemas allocating memoria => Mensaje de
           error
80     li          a0,FILE_DESCRIPTOR_STDERR
           # Cargo en a0 FILE_DESCRIPTOR_STDERR.
81     la          a1,MENSAJE_ERROR_MEMORIA_LEXICO
           # Cargo en a1 la direccion de memoria
           donde se encuentra el mensaje a cargar.
82     li          a2,
           BYTES_MENSAJE_ERROR_MEMORIA_LEXICO # Cargo en
           a2 la cantidad de bytes a escribir.
83     li          v0,SYS_write
84     syscall     # No controlo error porque
           sale de por si de la funcion por error.
85
86     li          v0,ERROR_MEMORY
87     sw          v0,32($fp) # Guardo el codigo de
           error (en v0) en la direccion de memoria 32($fp).
88     b           $ReturnLoadInLexico
           # Salto incondicional al return de la
           funcion (goto ReturnLoadInLexico).
89 $LoadCharacterInLexico:
90     # lexico[quantityCharacterInLexico] = character;
91     lw          v1,lexico
92     lw          v0,quantityCharacterInLexico
93     addu        v1,v1,v0 # Me corro en la memoria
           de lexico para guardar nuevo dato, guardo el dato
           en v1:
94     # lexico + quantityCharacterInLexico = lexico[
           quantityCharacterInLexico]
95     lbu         v0,24($fp) # Cargo en v0 el
           caracter a guardar (variable character).
96     sb          v0,0(v1) # Guardo en la direccion
           apuntada por v1 el caracter guardado en v0.
97
98     # quantityCharacterInLexico ++;
99     lw          v0,quantityCharacterInLexico
100    addu        v0,v0,1 # Incremento en 1 el
           valor de quantityCharacterInLexico.
101    sw          v0,quantityCharacterInLexico
102
103    # return OKEY;
104    sw          zero,32($fp) # Guardo OKEY (= 0) en
           la direccion de memoria 32($fp).
105 $ReturnLoadInLexico:
106    lw          v0,32($fp)
107    move        sp,$fp
108    lw          ra,48(sp)
109    lw          $fp,44(sp)

```



```

110      addu      sp,sp,56
111      j         ra
112      .end      loadInLexico
113
114
115  ## Mensajes de error
116
117      .rdata
118
119      .align    2
120  MENSAJE_ERROR_MEMORIA_LEXICO:
121      .ascii    "[Error] Hubo un error en memoria (lexico).
              \n\000"

```

Stack frame:

int loadInLexico(char character)				
Offset	Contents	Type reserved area	Comment	
48	ra	SRA		
44	fp		nothing to keep	
40	gp			
36		LTA	nothing to keep	
32	Resultado de la función		ERROR_MEMORY    OKEY	
28	bytesLexico			
24	character			
20			nothing to keep	
16		ABA	nothing to keep	
12	a3		Cada vez que se invoca a SYS_write (para informar errores), se guarda en a3 si hubo o no error.	
8	a2			
4	a1		Invocación a myRealloc: 1) lexico -> a0    bytesLexico -> a1	
0	a0		Invocación a mymalloc: 1) LEXICO_BUFFER_SIZE -> a0	
			Inicialmente contiene el valor del parametro character.	

Figura 7: Stack frame: loadInLexico

## 4.8. Código MIPS32: myfree.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #define MYMALLOC_SIGNATURE 0xdeadbeef
5
6
7  ##----- myfree -----##
8
9      .globl  myfree
10     .ent    myfree
11 myfree:
12     subu    sp, sp, 40
13     sw      ra, 32(sp)
14     sw      $fp, 28(sp)
15     sw      a0, 24(sp)  # Temporary: argument pointer.
16     sw      a0, 20(sp)  # Temporary: actual mmap(2)
17                     pointer.
18     move    $fp, sp
19
20     # Calculate the actual mmap(2) pointer.
21     #

```

```

21      lw      t0, 24(sp)
22      subu    t0, t0, 8
23      sw      t0, 20(sp)
24
25      # XXX Sanity check: the argument pointer must be
        checked
26      # in before we try to release the memory block.
27      #
28      # First, check the allocation signature.
29      #
30      lw      t0, 20(sp) # t0: actual mmap(2) pointer.
31      lw      t1, 0(t0)
32      bne     t1, MYMALLOC_SIGNATURE, myfree_die
33
34      # Second, check the memory block trailer.
35      #
36      lw      t0, 20(sp) # t0: actual mmap(2) pointer.
37      lw      t1, 4(t0)  # t1: actual mmap(2) block size.
38      addu    t2, t0, t1 # t2: trailer pointer.
39      lw      t3, -4(t2)
40      xor     t3, t3, t1
41      bne     t3, MYMALLOC_SIGNATURE, myfree_die
42
43      # All checks passed. Try to free this memory area.
44      #
45      li      v0, SYS_munmap
46      lw      a0, 20(sp) # a0: actual mmap(2) pointer.
47      lw      a1, 4(a0)  # a1: actual allocation size.
48      syscall
49
50      # Bail out if we cannot unmap this memory block.
51      #
52      bnez    v0, myfree_die
53
54      # Success.
55      #
56      j      myfree_return
57
58 myfree_die:
59      # Generate a segmentation fault by writing to the
        first
60      # byte of the address space (a.k.a. the NULL pointer
        ).
61      #
62      sw      t0, 0(zero)
63
64 myfree_return:
65      # Destroy the stack frame.
66      #
67      move    sp, $fp
68      lw      ra, 32(sp)
69      lw      $fp, 28(sp)
70      addu    sp, sp, 40
71

```

```

72         j        ra
73     .end      myfree

```

## 4.9. Código MIPS32: mymalloc.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #define MYMALLOC_SIGNATURE 0xdeadbeef
5
6  #ifndef PROT_READ
7  #define PROT_READ 0x01
8  #endif
9
10 #ifndef PROT_WRITE
11 #define PROT_WRITE 0x02
12 #endif
13
14 #ifndef MAP_PRIVATE
15 #define MAP_PRIVATE 0x02
16 #endif
17
18 #ifndef MAP_ANON
19 #define MAP_ANON 0x1000
20 #endif
21
22
23 ##----- mymalloc -----##
24
25     .text
26     .align 2
27     .globl mymalloc
28     .ent    mymalloc
29 mymalloc:
30     subu    sp, sp, 56
31     sw      ra, 48(sp)
32     sw      $fp, 44(sp)
33     sw      a0, 40(sp) # Temporary: original allocation
34                        size.
35     sw      a0, 36(sp) # Temporary: actual allocation
36                        size.
37     li      t0, -1
38     sw      t0, 32(sp) # Temporary: return value (
39                        defaults to -1).
40
41 #if 0
42     sw      a0, 28(sp) # Argument building area (#8?).
43     sw      a0, 24(sp) # Argument building area (#7?).
44     sw      a0, 20(sp) # Argument building area (#6).
45     sw      a0, 16(sp) # Argument building area (#5).
46     sw      a0, 12(sp) # Argument building area (#4, a3
47                        ).

```

```

43      sw      a0, 8(sp)  # Argument building area (#3, a2
44      ).
45      sw      a0, 4(sp)  # Argument building area (#2, a1
46      ).
47      sw      a0, 0(sp)  # Argument building area (#1, a0
48      ).
49  #endif
50      move     $fp, sp
51
52      # Adjust the original allocation size to a 4-byte
53      boundary.
54      #
55      lw      t0, 40(sp)
56      addiu   t0, t0, 3
57      and     t0, t0, 0xffffffffc
58      sw      t0, 40(sp)
59
60      # Increment the allocation size by 12 units, in
61      order to
62      # make room for the allocation signature, block size
63      and
64      # trailer information.
65      #
66      lw      t0, 40(sp)
67      addiu   t0, t0, 12
68      sw      t0, 36(sp)
69
70      # mmap(0, sz, PROT_READ|PROT_WRITE, MAP_PRIVATE|
71      MAP_ANON, -1, 0)
72      #
73      li      v0, SYS_mmap
74      li      a0, 0
75      lw      a1, 36(sp)
76      li      a2, PROT_READ|PROT_WRITE
77      li      a3, MAP_PRIVATE|MAP_ANON
78
79      # According to mmap(2), the file descriptor
80      # must be specified as -1 when using MAP_ANON.
81      #
82      li      t0, -1
83      sw      t0, 16(sp)
84
85      # Use a trivial offset.
86      #
87      li      t0, 0
88      sw      t0, 20(sp)
89
90      # XXX TODO.
91      #
92      sw      zero, 24(sp)
93      sw      zero, 28(sp)
94
95      # Execute the syscall, save the return value.
96      #

```

```

90      syscall
91      sw      v0, 32(sp)
92      beqz    v0, mymalloc_return
93
94      # Success. Check out the allocated pointer.
95      #
96      lw      t0, 32(sp)
97      li      t1, MYMALLOC_SIGNATURE
98      sw      t1, 0(t0)
99
100     # The actual allocation size goes right after the
101     # signature.
102     lw      t0, 32(sp)
103     lw      t1, 36(sp)
104     sw      t1, 4(t0)
105
106     # Trailer information.
107     #
108     lw      t0, 36(sp) # t0: actual allocation size.
109     lw      t1, 32(sp) # t1: Pointer.
110     addu    t1, t1, t0 # t1 now points to the trailing
111     4-byte area.
112     xor     t2, t0, MYMALLOC_SIGNATURE
113     sw      t2, -4(t1)
114
115     # Increment the result pointer.
116     #
117     lw      t0, 32(sp)
118     addiu   t0, t0, 8
119     sw      t0, 32(sp)
120
121     mymalloc_return:
122     # Restore the return value.
123     #
124     lw      v0, 32(sp)
125
126     # Destroy the stack frame.
127     #
128     move    sp, $fp
129     lw      ra, 48(sp)
130     lw      $fp, 44(sp)
131     addu    sp, sp, 56
132
133     j      ra
134     .end   mymalloc

```

#### 4.10. Código MIPS32: myRealloc.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3

```

```

4  #STATICS VAR DEFINITIONS
5
6  #define FALSE                0
7  #define TRUE                 1
8  #define DIR_NULL             0
9
10 ##----- myRealloc -----##
11
12     .align                    2
13     .globl                    myRealloc
14     .ent                      myRealloc
15 myRealloc:
16     .frame                    $fp,64,ra
17     .set                      noreorder
18     .cpload                   t9
19     .set                      reorder
20
21     #Stack frame creation
22     subu                      sp,sp,64
23
24     .cprestore 16
25     sw                        ra,56(sp)
26     sw                        $fp,52(sp)
27     sw                        gp,48(sp)
28     move                      $fp,sp
29
30     # Parameters
31     sw                        a0,64($fp) # Guardo en la direccion
32                                     de memoria 64($fp) la variable ptr (void * ptr).
33     sw                        a1,68($fp) # Guardo en la direccion
34                                     de memoria 68($fp) la variable tamanyoNew (
35                                     size_t tamanyoNew).
36     sw                        a2,72($fp) # Guardo en la direccion
37                                     de memoria 72($fp) la variable tamanyoOld (int
38                                     tamanyoOld).
39
40     lw                        v0,68($fp) # Cargo en v0 el
41                                     contenido de la variable tamanyoNew, que esta en
42                                     la direccion de memoria 68($fp)
43     bne                       v0,zero,
44                                     $MyReallocContinueValidations # If (tamanyoNew
45                                     != 0) goto MyReallocContinueValidations
46
47     # If (tamanyoNew == 0)
48     lw                        a0,64($fp) # Cargo en a0 la
49                                     direccion de memoria guardada en la direccion 64(
50                                     $fp), o sea, la variable * ptr.
51     la                        t9,myfree # Cargo la direccion de
52                                     la funcion myfree.
53     jal                      ra,t9 # Ejecuto la funcion myfree.
54     sw                        zero,64($fp) # Coloco el puntero
55                                     apuntando a NULL (ptr = NULL;).
56     sw                        zero,40($fp) # Coloco en la
57                                     direccion de memoria 40($fp) NULL, que seria el

```

```

44         resultado de la funcion myRealloc.
        b           $MyReallocReturn
           # Salto incondicional para retornar
           resultado de myRealloc.
45 $MyReallocContinueValidations:
46     lw           a0,68($fp) # Cargo en a0 el
           contenido guardado en la direccion 68($fp), o sea
           , la variable tamanyoNew.
47     la           t9,mymalloc # Cargo la direccion de
           la funcion mymalloc.
48     jal          ra,t9      # Ejecuto la funcion
           mymalloc.
49     sw           v0,24($fp) # Guardo en la direccion
           24($fp) el contenido de v0, que seria la
           direccion de la memoria asignada con mymalloc.
50     lw           v0,24($fp) # Cargo en v0 la
           direccion de la memoria asignada con mymalloc (
           void * ptrNew = (void *) mymalloc(tamanyoNew);).
51
52     # (ptrNew == NULL) ?
53     bne          v0,DIR_NULL,
           $MyReallocContinueValidationsWithMemory # If (
           ptrNew != NULL) goto
           MyReallocContinueValidationsWithMemory
54     sw           zero,40($fp) # Coloco en la
           direccion de memoria 40($fp) NULL, que seria el
           resultado de la funcion myRealloc.
55     b           $MyReallocReturn
           # Salto incondicional para retornar
           resultado de myRealloc.
56 $MyReallocContinueValidationsWithMemory:
57     lw           v0,64($fp) # Cargo en v0 la
           direccion de memoria guardada en la direccion 64(
           $fp), o sea, la variable * ptr.
58     bne          v0,DIR_NULL,
           $MyReallocContinueWithLoadCharacters # If (ptr !=
           NULL) goto MyReallocContinueWithLoadCharacters
59
60     # (ptr == NULL) ?
61     lw           v0,24($fp) # Cargo en v0 la
           direccion de memoria guardada en la direccion 24(
           $fp), o sea, la variable * ptrNew,
62     # que seria la direccion de la memoria asignada con
           mymalloc.
63     sw           v0,40($fp) # Coloco en la direccion
           de memoria 40($fp) el contenido de v0 (* ptrNew)
           , que seria el resultado de la funcion myRealloc.
64     b           $MyReallocReturn
           # Salto incondicional para retornar
           resultado de myRealloc.
65 $MyReallocContinueWithLoadCharacters:
66     lw           v0,68($fp) # Cargo en v0 el
           contenido guardado en la direccion 68($fp), o sea
           , la variable tamanyoNew.

```

```

67      sw          v0,28($fp) # Guardo en la direccion
      de memoria 28($fp) la variable tamanyoNew
      guardada en v0 (int end = tamanyoNew;).
68
69      lw          v1,72($fp) # Cargo en v1 el
      contenido guardado en la direccion 72($fp), o sea
      , la variable tamanyoOld.
70      lw          v0,68($fp) # Cargo en v0 el
      contenido guardado en la direccion 68($fp), o sea
      , la variable tamanyoNew, para poder luego hacer
      comparacion.
71
72      # (tamanyoOld < tamanyoNew) ?
73      sltu        v0,v1,v0 # Compara el contenido de
      la variable tamanyoOld (v1) con tamanyoNew (v0),
      y guarda true en v0 si
74      # el primero (tamanyoOld) es mas chico que el segundo (
      tamanyoNew).
75      beq         v0,FALSE,$MyReallocLoadCharacters
      # If (tamanyoOld >= tamanyoNew) goto
      MyReallocLoadCharacters
76      lw          v0,72($fp) # Cargo en v0 el
      contenido guardado en la direccion 72($fp), o sea
      , la variable tamanyoOld.
77      sw          v0,28($fp) # Guardo en la direccion
      28($fp), que seria la variable end, el contenido
      de la variable tamanyoOld (end = tamanyoOld;).
78      $MyReallocLoadCharacters:
79      lw          v0,24($fp) # Cargo en v0 el
      contenido guardado en la direccion 24($fp), o sea
      , la variable ptrNew.
80      sw          v0,32($fp) # Guardo en la direccion
      de memoria 24($fp) el contenido de v0 (char *tmp
      = ptrNew;).
81      lw          v0,64($fp) # Cargo en v0 el
      contenido guardado en la direccion 64($fp), o sea
      , la variable ptr.
82      sw          v0,36($fp) # Guardo en la direccion
      de memoria 36($fp) el contenido de v0 (const
      char *src = ptr;).
83      $MyReallocWhileLoadCharacter:
84      lw          v0,28($fp) # Cargo en v0 el
      contenido guardado en la direccion 28($fp), o sea
      , la variable end.
85      addu        v0,v0,-1 # Decremento en 1 el
      contenido de v0 (end --).
86      move        v1,v0 # Muevo el contenido de v0 a
      v1.
87      sw          v1,28($fp) # Guardo en la direccion
      de memoria 28($fp), que seria en donde estaba
      end, el nuevo valor de end (habia sido
      decrementado en 1).
88      li          v0,-1 # Cargo en v0 el literal -1.

```



```

89      bne          v1,v0,$MyReallocContinueWhileLoad
          # If ( end != -1) goto
          MyReallocContinueWhileLoad.
90      b           $MyReallocFinalizedWhileLoad
          # Salto incondicional fuera del while,
          porque la variable end es -1.
91 $MyReallocContinueWhileLoad:
92      # *tmp = *src;
93      lw          v1,32($fp) # Cargo en v1 el
          contenido guardado en la direccion 32($fp), que
          seria *tmp.
94      lw          v0,36($fp) # Cargo en v0 el
          contenido guardado en la direccion 36($fp), que
          seria *src.
95      lbu         v0,0(v0) # Cargo la direccion de
          memoria en v0 de src.
96      sb          v0,0(v1) # Guardo en la direccion
          apuntada por el contenido de v1, la direccion de
          memoria guardada en v0 (*tmp = *src;).
97
98      # tmp ++
99      lw          v0,32($fp) # Cargo en v0 el
          contenido guardado en la direccion 32($fp), que
          seria *tmp.
100     addu         v0,v0,1 # Incremento en 1 el
          contenido guardado en v0 (tmp ++).
101     sw          v0,32($fp) # Guardo en la direccion
          de memoria 32($fp) lo que tenia v0 (el resultado
          de hacer tmp ++).
102
103     # src ++
104     lw          v0,36($fp) # Cargo en v0 el
          contenido guardado en la direccion 36($fp), que
          seria *src.
105     addu         v0,v0,1 # Incremento en 1 el
          contenido guardado en v0 (src ++).
106     sw          v0,36($fp) # Guardo en la direccion
          de memoria 36($fp) lo que tenia v0 (el resultado
          de hacer src ++).
107
108     b           $MyReallocWhileLoadCharacter
          # Vuelvo a entrar al while
109 $MyReallocFinalizedWhileLoad:
110     lw          a0,64($fp) # Cargo en v0 el
          contenido guardado en la direccion 64($fp), que
          seria *ptr.
111     la          t9,myfree # Cargo la direccion de
          la funcion myfree.
112     jal         ra,t9 # Ejecuto la funcion myfree.
113     sw          zero,64($fp) # Coloco el puntero
          apuntando a NULL (ptr = NULL;).
114
115     lw          v0,24($fp) # Cargo en v0 la
          direccion de memoria guardada en la direccion 24(

```

```

116     $fp), o sea, la variable * ptrNew, que seria
117     # la direccion de la memoria asignada con mymalloc..
118     sw          v0,40($fp) # Guardo en la direccion
119     de memoria 40($fp) el contenido de v0 (* ptrNew)
120     , que seria el resultado de la funcion myRealloc.
121 $MyReallocReturn:
122     lw          v0,40($fp) # Cargo en v0 el
123     resultado de la funcion myRealloc guardado en la
124     direccion de memoria 40($fp).
125     move        sp,$fp
126     lw          ra,56(sp)
127     lw          $fp,52(sp)
128     addu        sp,sp,64
129     j           ra      # Jump and return
130     .end        myRealloc

```

Stack frame:

void * myRealloc(void * ptr, size_t tamanyoNew, int tamanyoOld)				
Offset	Contents	Type reserved area	Comment	
72	tamanyoOld			
68	tamanyoNew			
64	* ptr			
60			nothing to keep	
56	ra	SRA		
52	fp			
48	gp			
44				
40	Resultado de la función	LTA	nothing to keep	
36	* src		NULL    *ptrNew	
32	* tmp		ptr    tmp	
28	end		ptrNew    src	
24	* ptrNew		tamanyoNew    tamanyoOld	
20				
16			nothing to keep	
12	a3	ABA	nothing to keep	
8	a2			Invocación a myfree: 1) *ptr -> a0
4	a1		Inicialmente contiene el valor del parametro * buffer.	
0	a0		Inicialmente contiene el valor del parametro * amountSavedInOBuffer.	Invocación a mymalloc: 1) tamanyoNew -> a0

Figura 8: Stack frame: myRealloc

## 4.11. Código MIPS32: palindrome.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FALSE          0
7  #define TRUE           1
8  #define DIR_NULL      0
9  #define FILE_DESCRIPTOR_STDERR  2
10
11 # Resultados de funciones posibles
12 #define OKEY            0
13 #define ERROR_MEMORY    2
14 #define ERROR_READ      3
15 #define ERROR_WRITE     4

```

```

16 #define LOAD_I_BUFFER          5
17 #define OKEY_I_FILE            -4
18
19 # Size mensajes
20 #define BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER    60
21 #define BYTES_MENSAJE_ERROR_MEMORIA_IBUFFER    60
22 #define BYTES_MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED 64
23 #define BYTES_MENSAJE_ERROR_Lectura_ARCHIVO    60
24
25
26 ##----- palindrome -----##
27
28
29 .align          2
30 .globl          palindrome
31 .ent            palindrome
32 palindrome:
33 .frame          $fp,64,ra
34 .set            noreorder
35 .cpload         t9
36 .set            reorder
37
38 #Stack frame creation
39 subu            sp,sp,64
40
41 .cpstore 16
42 sw              ra,56(sp)
43 sw              $fp,52(sp)
44 sw              gp,48(sp)
45 move            $fp,sp
46
47 # Parameters
48 sw              a0,64($fp) # Guardo en la direccion
49                  de memoria 64($fp) la variable ifd (int ifd).
50 sw              a1,68($fp) # Guardo en la direccion
51                  de memoria 68($fp) la variable ibytes (size_t
52                  ibytes).
53 sw              a2,72($fp) # Guardo en la direccion
54                  de memoria 72($fp) la variable ofd (int ofd).
55 sw              a3,76($fp) # Guardo en la direccion
56                  de memoria 76($fp) la variable obytes (size_t
57                  obytes).
58
59 # isize = ibytes;
60 lw              v0,68($fp) # Cargo en v0 ibytes,
61                  guardado en 68($fp).
62 sw              v0,isize # Guardo en isize ibytes.
63
64 # osize = obytes;
65 lw              v0,76($fp) # Cargo en v0 obytes,
66                  guardado en 76($fp).
67 sw              v0,osize # Guardo en osize obytes.
68
69 # oFileDescriptor = ofd;

```

```

62      lw          v0,72($fp) # Cargo en v0 ofd,
        guardado en 72($fp).
63      sw          v0,oFileDescriptor
        # Guardo en oFileDescriptor ofd.
64
65      # ibuffer = loadBufferInitial(isize, ibuffer);
66      lw          a0, isize # Cargo en a0 isize.
        Parametro de la funcion loadBufferInitial.
67      lw          a1, ibuffer # Cargo en a1 ibuffer.
        Parametro de la funcion loadBufferInitial.
68      la          t9, loadBufferInitial
69      jal         ra, t9 # Ejecuto la funcion
        loadBufferInitial.
70      sw          v0, ibuffer # Asigno a ibuffer el
        resultado de la funcion loadBufferInitial.
71
72      # (ibuffer == NULL) ?
73      lw          v0, ibuffer
74      bne         v0, DIR_NULL, $LoadOBuffer
        # If (ibuffer != NULL) goto LoadOBuffer
75
76      # ibuffer is NULL
77      li          v0, ERROR_MEMORY
78      sw          v0, 44($fp) # Guardo el codigo de
        error en la direccion 44($fp).
79      b           $ReturnPalindrome
        # Salto incondicional al return de la
        funcion.
80 $LoadOBuffer:
81      # obuffer = loadBufferInitial(usize, obuffer);
82      lw          a0, usize
83      lw          a1, obuffer
84      la          t9, loadBufferInitial
85      jal         ra, t9
86      sw          v0, obuffer
87
88      # (obuffer == NULL) ?
89      lw          v0, obuffer
90      bne         v0, DIR_NULL,
        $LoadAmountSavedInOBuffer # If (obuffer != NULL
        ) goto LoadAmountSavedInOBuffer
91
92      # free(ibuffer); ibuffer = NULL;
93      lw          a0, ibuffer
94      la          t9, myfree
95      jal         ra, t9
96      sw          zero, ibuffer
97
98      li          v0, ERROR_MEMORY
99      sw          v0, 44($fp) # Guardo el codigo de
        error en la direccion 44($fp).
100     b           $ReturnPalindrome
        # Salto incondicional al return de la
        funcion.

```

```

101 $LoadAmountSavedInOBuffer:
102     li            a0,4      # Cargo en a0 la cantidad de
                                bytes a asignar (por ser un int, son 4 bytes).
103     la            t9,mymalloc
104     jal           ra,t9
105     sw            v0,24($fp) # En v0 esta el
                                resultado de mymalloc. Asigno este resultado a la
                                direccion
106     # 24($fp), que representaria a la variable *
                                amountSavedInOBuffer.
107
108     # amountSavedInOBuffer == NULL ?
109     lw            v0,24($fp)
110     bne           v0,DIR_NULL,
                                $ContinueProcessToLoadIBuffer # If (
                                amountSavedInOBuffer != NULL) goto
                                ContinueProcessToLoadIBuffer
111
112     # amountSavedInOBuffer is NULL => Mensaje de error
113     li            a0,FILE_DESCRIPTOR_STDERR
                                # Cargo en a0 FILE_DESCRIPTOR_STDERR.
114     la            a1,
                                MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED # Cargo en
                                a1 la direccion de memoria donde se encuentra el
                                mensaje a cargar.
115     li            a2,
                                BYTES_MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED # Cargo
                                en a2 la cantidad de bytes a escribir.
116     li            v0, SYS_write
117     syscall       # No controlo error porque
                                sale de por si de la funcion por error.
118
119     # myfree(ibuffer)
120     lw            a0,ibuffer # Cargo en a0 ibuffer.
                                Parametro de la funcion myfree.
121     la            t9,myfree # Cargo en t9 la
                                direccion de la funcion myfree.
122     jal           ra,t9      # Ejecuto la funcion myfree.
123     sw            zero,ibuffer # Asigno NULL a
                                ibuffer.
124
125     # myfree(obuffer)
126     lw            a0,obuffer # Cargo en a0 obuffer.
                                Parametro de la funcion myfree.
127     la            t9,myfree # Cargo en t9 la
                                direccion de la funcion myfree.
128     jal           ra,t9      # Ejecuto la funcion myfree.
129     sw            zero,obuffer # Asigno NULL a
                                obuffer.
130
131     li            v0,ERROR_MEMORY # Cargo en v0 el
                                codigo de error.
132     sw            v0,44($fp)
133     b             $ReturnPalindrome

```

```

134 $ContinueProcessToLoadIBuffer:
135     # amountSavedInOBuffer[0] = 0;
136     lw          v0,24($fp)
137     sw          zero,0(v0)
138
139     # int rdoProcess = OKEY;
140     sw          zero,28($fp)
141
142     # int error = FALSE;
143     sw          zero,32($fp)
144
145     # int rdoLoadIBuffer = OKEY_I_FILE;
146     li          v0,OKEY_I_FILE
147     sw          v0,36($fp)
148 $WhilePalindrome:
149     # (rdoLoadIBuffer == OKEY_I_FILE) ?
150     lw          v1,36($fp)
151     li          v0,OKEY_I_FILE
152     bne         v1,v0,$CleanBuffersSincePalindrome
153         # If (rdoLoadIBuffer != OKEY_I_FILE) goto
154         CleanBuffersSincePalindrome
155
156     # (error == FALSE) ?
157     lw          v0,32($fp)
158     bne         v0,FALSE,
159         $CleanBuffersSincePalindrome # If (error !=
160         FALSE) goto CleanBuffersSincePalindrome
161
162     # rdoLoadIBuffer = loadIBufferWithIFile(ibytes, ifd)
163     ;
164     lw          a0,68($fp)
165     lw          a1,64($fp)
166     la          t9,loadIBufferWithIFile
167     jal         ra,t9
168     sw          v0,36($fp) # Cargo en la direccion
169     36($fp) el resultado de la funcion
170     loadIBufferWithIFile,
171     # que representaria a la variable
172     rdoLoadIBuffer.
173
174     # (ibuffer != NULL && ibuffer[0] != '\0') ?
175
176     # (ibuffer != NULL) ?
177     lw          v0,ibuffer
178     beq         v0,DIR_NULL,$WhilePalindrome
179         # If (ibuffer == NULL) goto
180         WhilePalindrome
181
182     # (ibuffer[0] != '\0') ?
183     lw          v0,ibuffer
184     lb          v0,0(v0)
185     beq         v0,zero,$WhilePalindrome
186         # If (ibuffer[0] == '\0') goto
187         WhilePalindrome

```

```

176      # int resultProcessWrite = processDataInIBuffer(
177      #       ibuffer, amountSavedInOBuffer);
178      lw      a0, ibuffer
179      lw      a1, 24($fp)
180      la      t9, processDataInIBuffer
181      jal     ra, t9
182      sw      v0, 40($fp) # En la direccion 40($fp
        ) guarda el resultado de la funcion
        processDataInIBuffer,
183      # que representa la variable resultProcessWrite.
184
185      # (resultProcessWrite == LOAD_I_BUFFER) ?
186      lw      v1, 40($fp)
187      li      v0, LOAD_I_BUFFER
188      bne     v1, v0,
        $ContinueValidationResultProcessDataInIBuffer #
        If (resultProcessWrite != LOAD_I_BUFFER) goto
        ContinueValidationResultProcessDataInIBuffer.
189
190      # resultProcessWrite is equal LOAD_I_BUFFER
191
192      # initializeBuffer(abytes, ibuffer);
193      lw      a0, 68($fp)
194      lw      a1, ibuffer
195      la      t9, initializeBuffer
196      jal     ra, t9 # Ejecuto la funcion
        initializeBuffer.
197      $ContinueValidationResultProcessDataInIBuffer:
198      # (resultProcessWrite == ERROR_MEMORY ||
        resultProcessWrite == ERROR_WRITE) ?
199
200      # resultProcessWrite == ERROR_MEMORY ?
201      lw      v1, 40($fp) # Cargo en v1
        resultProcessWrite.
202      li      v0, ERROR_MEMORY
203      beq     v1, v0, $LoadErrorOfExecutePalinWrite
        # If (resultProcessWrite == ERROR_MEMORY)
        goto LoadErrorOfExecutePalinWrite.
204
205      # resultProcessWrite is not equal ERROR_MEMORY
206
207      # resultProcessWrite == ERROR_WRITE ?
208      lw      v1, 40($fp) # Cargo en v1
        resultProcessWrite.
209      li      v0, ERROR_WRITE
210      beq     v1, v0, $LoadErrorOfExecutePalinWrite
        # If (resultProcessWrite == ERROR_WRITE) goto
        LoadErrorOfExecutePalinWrite.
211
212      # No hay errores
213      b      $WhilePalindrome
214      $LoadErrorOfExecutePalinWrite:
215      li      v0, TRUE

```

```

216         sw             v0,32($fp) # Asigno a la variable
                error TRUE.
217
218         # rdoProcess = resultProcessWrite;
219         lw             v0,40($fp) # Cargo en v0
                resultProcessWrite.
220         sw             v0,28($fp) # Asigno a la variable
                rdoProcess resultProcessWrite.
221
222         b             $WhilePalindrome
                # Vuelvo a intentar entrar al loop.
223 $CleanBuffersSincePalindrome:
224         # int rdoClean = cleanBuffers(amountSavedIn0Buffer);
225         lw             a0,24($fp) # Cargo en a0
                amountSavedIn0Buffer. Parametro de la funcion
                cleanBuffers.
226         la             t9,cleanBuffers
227         jal            ra,t9 # Ejecuto la funcion
                cleanBuffers.
228         sw             v0,40($fp) # Guardo en la direccion
                40($fp) el resultado de ejecutar la funcion
                cleanBuffers. Seria rdoClean.
229
230         lw             v0,24($fp) # Cargo en v0
                amountSavedIn0Buffer.
231         beq            v0,DIR_NULL,$VerifyResultClean
                # if (amountSavedIn0Buffer == NULL) goto
                VerifyResultClean.
232
233         # free(amountSavedIn0Buffer); amountSavedIn0Buffer =
                NULL;
234         lw             a0,24($fp)
235         la             t9,myfree
236         jal            ra,t9
237         sw             zero,24($fp)
238 $VerifyResultClean:
239         # (rdoClean != OKEY) ?
240         lw             v0,40($fp) # Cargo en v0 rdoClean.
241         beq            v0,OKEY,$ReturnResultProcess
                # if (rdoClean == OKEY) goto
                ReturnResultProcess
242         lw             v0,40($fp)
243         sw             v0,44($fp) # Cargo en la direccion
                44($fp) rdoClean
244         b             $ReturnPalindrome
245 $ReturnResultProcess:
246         lw             v0,28($fp)
247         sw             v0,44($fp) # Cargo en la direccion
                44($fp) rdoProcess
248 $ReturnPalindrome:
249         lw             v0,44($fp)
250         move            sp,$fp
251         lw             ra,56(sp)
252         lw             $fp,52(sp)

```



```

253         addu            sp,sp,64
254         j              ra
255         .end            palindrome
256
257
258     ## Variables auxiliares
259
260         .data
261
262         .globl  isize
263         .align  2
264     isize:
265         .space  4
266
267         .globl  osize
268         .align  2
269     osize:
270         .space  4
271
272         .globl  oFileDescriptor
273         .align  2
274     oFileDescriptor:
275         .space  4
276
277         .globl          lexico
278         .section        .bss
279         .align  2
280         .type  lexico, @object
281         .size  lexico, 4
282     lexico:
283         .space  4
284
285         .globl  quantityCharacterInLexico
286         .align  2
287         .type  quantityCharacterInLexico, @object
288         .size  quantityCharacterInLexico, 4
289     quantityCharacterInLexico:
290         .space  4
291
292         .globl  savedInOFile
293         .align  2
294         .type  savedInOFile, @object
295         .size  savedInOFile, 4
296     savedInOFile:
297         .space  4
298
299         .globl  obuffer
300         .align  2
301         .type  obuffer, @object
302         .size  obuffer, 4
303     obuffer:
304         .space  4
305
306         .globl  bytesLexico

```

```

307         .align 2
308         .type bytesLexico, @object
309         .size bytesLexico, 4
310 bytesLexico:
311         .space 4
312
313         .globl ibuffer
314         .align 2
315         .type ibuffer, @object
316         .size ibuffer, 4
317 ibuffer:
318         .space 4
319
320
321
322 ## Mensajes de error
323
324         .rdata
325
326         .align 2
327 MENSAJE_ERROR_MEMORIA_OBUFFER:
328         .ascii "[Error] Hubo un error de asignacion de
329             memoria (obuffer)"
330         .ascii ". \n\000"
331
332         .align 2
333 MENSAJE_ERROR_MEMORIA_IBUFFER:
334         .ascii "[Error] Hubo un error de asignacion de
335             memoria (ibuffer)"
336         .ascii ". \n\000"
337
338         .align 2
339 MENSAJE_ERROR_MEMORIA_AMOUNT_SAVED:
340         .ascii "[Error] Hubo un error de asignacion de
341             memoria (amountSa"
342         .ascii "ved). \n\000"
343
344         .align 2
345 MENSAJE_ERROR_LECTURA_ARCHIVO:
346         .ascii "[Error] Hubo un error en la lectura de
347             datos del archivo"
348         .ascii ". \n\000"

```

Stack frame:

Figura 9: Stack frame: palindrome

Figura 10: Stack frame: palindrome

#### 4.12. Código MIPS32: processDataInIBuffer.S

```

1
2 #include <mips/regdef.h>
3 #include <sys/syscall.h>
4
5 #STATICS VAR DEFINITIONS
6
7 #define FALSE                0
8 #define TRUE                 1
9 #define DIR_NULL             0
10 #define FILE_DESCRIPTOR_STDERR 2
11 #define LINE_BREAK           10
12
13 # Resultados de funciones posibles
14 #define OKEY                  0
15 #define ERROR_MEMORY          2
16 #define LOAD_I_BUFFER         5
17
18 # Size mensajes
19 #define BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER 60
20
21
22 ##----- processDataInIBuffer -----##
23
24     .align                2
25     .globl                processDataInIBuffer
26     .ent                  processDataInIBuffer
27 processDataInIBuffer:
28     .frame                $fp,80,ra
29     .set                  noreorder
30     .cpload               t9
31     .set                  reorder
32
33     #Stack frame creation
34     subu                  sp,sp,80
35
36     .cpstore 16
37     sw                    ra,72(sp)
38     sw                    $fp,68(sp)
39     sw                    gp,64(sp)
40     move                  $fp,sp
41
42     # Parameter
43     sw                    a0,80($fp) # Guardo en la direccion
44                                     de memoria 80($fp) la variable * ibuffer (char *
45                                     ibuffer).
46     sw                    a1,84($fp) # Guardo en la direccion
47                                     de memoria 84($fp) la variable *
48                                     amountSavedInOBuffer (int * amountSavedInOBuffer)
49                                     .
50
51     sw                    zero,24($fp) # Guardo en la
52                                     direccion de memoria 24($fp) el valor FALSE (que
53                                     seria 0), representa la variable findEnd (int

```

```

findEnd = FALSE).
47      sw          zero,28($fp) # Guardo en la
      direccion de memoria 28($fp) el valor FALSE (que
      seria 0), representa la variable
48 loadIBuffer (int loadIBuffer = FALSE).
49      sw          zero,32($fp) # Guardo en la
      direccion de memoria 32($fp) el valor 0,
      representa la variable idx (int idx = 0).
50      sw          zero,36($fp) # Guardo en la
      direccion de memoria 36($fp) el valor OKEY (que
      seria 0), representa la variable rdo (int rdo =
      FALSE).
51 $WhileProcessDataInIBuffer:
52     # findEnd == FALSE ?
53     lw          v0,24($fp) # Cargo en v0 el
      contenido de la direccion de memoria 24($fp), que
      seria la variable findEnd.
54     bne         v0,FALSE,
      $LeaveWhileProcessDataInIBuffer # If (findEnd !=
      FALSE) goto LeaveWhileProcessDataInIBuffer.
55
56     # loadIBuffer == FALSE ?
57     lw          v0,28($fp) # Cargo en v0 el
      contenido de la direccion de memoria 28($fp), que
      seria la variable loadIBuffer.
58     bne         v0,zero,
      $LeaveWhileProcessDataInIBuffer # If (loadIBuffer
      != FALSE) goto LeaveWhileProcessDataInIBuffer.
59
60     # Comienzo a ejecutar las intruccioness dentro del
      while
61
62     # char character = ibuffer[idx];
63     lw          v1,80($fp) # Cargo en v1 lo
      guardado en la direccion de memoria 80($fp), que
      seria la variable ibuffer.
64     lw          v0,32($fp) # Cargo en v0 lo
      guardado en la direccion de memoria 32($fp), que
      seria la variable idx.
65     addu        v0,v1,v0    # Me corro en la
      direccion de memoria: a la apuntada por v1 (
      ibuffer) me corro la cantidad de posiciones
      establecidas por idx.
66     lbu         v0,0(v0)    # Cargo en v0 la
      direccion de memoria guardada en v0 (calculada en
      el paso anterior).
67     sb          v0,40($fp) # Guardo en la direccion
      40($fp) lo guardado en v0. Representaria la
      variable character.
68
69     # character == '\0' ?
70     lb          v0,40($fp) # Cargo en v0 el
      contenido de la direccion de memoria 40($fp), que
      seria la variable character.

```

```

71      bne          v0,zero,
          $VerifyCharacterToLoadInLexico # If (character
          != '\0') goto VerifyCharacterToLoadInLexico.
72
73      # character is equal '\0'
74      li          v0,TRUE      # Cargo en v0 TRUE (que
          seria el literal 1).
75      sw          v0,24($fp) # Guardo en la direccion
          de memoria 24($fp) el contenido de v0 (findEnd =
          TRUE).
76  $VerifyCharacterToLoadInLexico:
77      # findEnd != TRUE
78      lw          v1,24($fp) # Cargo en v1 el
          contenido de la direccion de memoria 24($fp), que
          seria findEnd.
79      li          v0,TRUE      # Cargo en v0 el literal
          1 (TRUE) para hacer luego una comparacion.
80
81      # Si findEnd (v1) es igual a TRUE (v0), salto a
          VerifyQuantityCharacterInLexico. findEnd es igual
          a TRUE. Continuo validaciones para cargar
          caracter en lexico. Voy a verificar si el
          caracter es una keyword.
82      beq         v1,v0,
          $VerifyQuantityCharacterInLexico
83      lb          v0,40($fp) # Cargo en v0 el
          contenido en la direccion de memoria 40($fp), que
          seria character.
84      move        a0,v0 # Muevo el contenido de v0 a
          a0. Voy a pasar como parametro la variable
          character a la funcion isKeywords.
85      la          t9,isKeywords # Cargo la direccion
          de memoria de isKeywords.
86      jal         ra,t9 # Ejecuto la funcion
          isKeywords.
87      move        v1,v0 # Muevo el resultado de la
          funcion isKeywords, que esta en v0, a v1.
88      li          v0,TRUE
89      bne         v1,v0,
          $VerifyQuantityCharacterInLexico # Si el
          resultado de la funcion isKeywords (v1) no es
          igual a TRUE (v0), salto a
          VerifyQuantityCharacterInLexico.
90      # El caracter es una keyword.
91      lb          v0,40($fp) # Cargo en v0 el
          contenido en la direccion de memoria 40($fp), que
          seria character.
92      move        a0,v0 # Muevo el contenido de v0 a
          a0. Parametro de la funcion loadInLexico.
93      la          t9,loadInLexico
94      jal         ra,t9 # Ejecuto loadInLexico.
95      sw          v0,44($fp) # Guardo el resultado de
          loadInLexico en la direccion 44($fp). Variable
          rdo.

```

```

96         lw          v0,44($fp)
97         beq         v0,OKEY,$VerifyLoadIBuffer # If (
           rdo == OKEY) goto VerifyLoadIBuffer.
98
99         # rdo != OKEY
100        lw          v0,44($fp) # Cargo v0 el codigo de
           error.
101        sw          v0,56($fp) # Guardo en la direccion
           56($fp) el codigo de error.
102        b          $ReturnProcessDataInIBuffer # Salto
           incondicional al return de la funcion.
103        $VerifyQuantityCharacterInLexico:
104        # quantityCharacterInLexico > 0 ?
105        lw          v0,quantityCharacterInLexico #
           Cargo en v0 quantityCharacterInLexico.
106        blez        v0,$VerifyLoadIBuffer # Si
           quantityCharacterInLexico (v0) es menor que 0,
           salto a VerifyLoadIBuffer.
107
108        # quantityCharacterInLexico > 0 => Verifico si
           lexico es palindromo.
109        lw          a0,lexico #
           Cargo en a0 lexico (parametro para la funcion
           verifyPalindromic).
110        lw          a1,quantityCharacterInLexico #
           Cargo en a1 quantityCharacterInLexico (parametro
           para la funcion verifyPalindromic).
111        la          t9,verifyPalindromic #
           Cargo en t9 la direccion de la verifyPalindromic.
112        jal         ra,t9 # Ejecuto la funcion
           verifyPalindromic.
113        sw          v0,44($fp) #
           Guardo en la direccion de memoria 44($fp) el
           resultado de la funcion verifyPalindromic, que
           esta
114               # almacenado en v0 y que
           representaria a la variable
           itsPalindromic.
115        lw          v1,44($fp) #
           Cargo en v1 lo que se encuentra en la direccion
           de memoria 44($fp) que seria el resultado
           # de la funcion verifyPalindromic.
116        li          v0,TRUE
117        bne         v1,v0,$myfreeLexico # If
           (itsPalindromic != TRUE) goto myfreeLexico.
118
119        # itsPalindromic is TRUE
120
121        # loadInLexico('\n');
122
123
124        li          a0,LINE_BREAK
125        la          t9,loadInLexico
126        jal         ra,t9 # Ejecuto loadInLexico.
           Agrego el salto de linea al lexico.

```

```

127      # int amountToSaved = (*amountSavedIn0Buffer) +
128      quantityCharacterInLexico;
129      lw      v0,84($fp) # Cargo en v0 lo almacenado en
la direccion de memoria 84($fp), que seria la
variable *amountSavedIn0Buffer.
130      lw      v1,0(v0) # Cargo lo almacenado en la
direccion de memoria apuntada por *
amountSavedIn0Buffer en v1.
131      lw      v0,quantityCharacterInLexico #
Cargo en v0 quantityCharacterInLexico.
132      addu    v0,v1,v0 # Hago (*amountSavedIn0Buffer) +
quantityCharacterInLexico, y guardo el resultado
en v0.
133      sw      v0,48($fp) # Guardo el resultado de la suma
(almacenado el v0) en la direccion de memoria
48($fp), que
134      # representaria a la variable
amountToSaved.
135
136      # (*amountSavedIn0Buffer) > 0 ?
137      lw      v0,84($fp) # Cargo en v0 lo almacenado en
la direccion de memoria 84($fp), que seria la
variable *amountSavedIn0Buffer.
138      lw      v0,0(v0) # Cargo lo almacenado en la
direccion de memoria apuntada por *
amountSavedIn0Buffer en v0.
139      bgtz    v0,$IncrementAmountToSaved # Si
el contenido de lo almacenado en la direccion
apuntada por amountSavedIn0Buffer (que esta en v0
)
140      # es mas grande que 0, salto a IncrementAmountToSaved.
141
142      # (*amountSavedIn0Buffer) <= 0
143      # savedIn0File == TRUE ?
144      lw      v1,savedIn0File # Cargo en v1 savedIn0File.
145      li      v0,TRUE
146      beq     v1,v0,$IncrementAmountToSaved # If
(savedIn0File == TRUE) goto
IncrementAmountToSaved.
147      b      $ContinueVerificationAboutAmountToSaved #
Salto incondicional a
ContinueVerificationAboutAmountToSaved.
148      $IncrementAmountToSaved:
149      # amountToSaved ++; Es para el separador entre
lexicos
150      lw      v0,48($fp) # Cargo en v0 lo guardado en la
direccion de memoria 48($fp), que representaria a
la variable amountToSaved.
151      addu    v0,v0,1 # Incremento en uno a
amountToSaved.
152      sw      v0,48($fp) # Guardo el nuevo valor de
amountToSaved (almacenado en v0) en la direccion
de memoria 48($fp).

```

```

153 $ContinueVerificationAboutAmountToSaved:
154     # amountToSaved > osize ?
155     lw      v0,48($fp) # Cargo en v0 lo guardado en la
                        # direccion de memoria 48($fp), que representaria a
                        # la variable amountToSaved.
156     lw      v1,osize # Cargo en v1 osize.
157     sltu    v0,v1,v0 # Si v1 (osize) es mas chico que
                        # v0 (amountToSaved), guardo TRUE en v0, sino
                        # guardo FALSE.
158     beq     v0,FALSE,$LoadLexicoInOBuffer # Si
                        # el resultado de la comparacion es FALSE (
                        # amountToSaved <= osize), salto a
                        # LoadLexicoInOBuffer.
159
160     # amountToSaved > osize
161     # Tomo la decision de pedir mas memoria para bajar
                        # el lexico completo
162     # y luego rearmo el buffer de salida y reinicio la
                        # cantidad guardada en 0.
163
164     # obuffer = myRealloc(obuffer, amountToSaved*sizeof(
                        # char), (*amountSavedInOBuffer));
165     lw      v0,84($fp) # Cargo en v0 lo guardado en la
                        # direccion de memoria 84($fp), que representaria a
                        # la variable *amountSavedInOBuffer.
166     lw      a0,obuffer # Cargo en a0 obuffer (parametro
                        # para la funcion myRealloc).
167     lw      a1,48($fp) # Cargo en a1 lo guardado en la
                        # direccion de memoria 48($fp), que representaria a
                        # la variable *amountToSaved. Parametro
168     # para la funcion myRealloc.
169     lw      a2,0(v0) # Cargo en a2 lo almacenado en la
                        # direccion de memoria guardada en v0 (parametro
                        # para la funcion myRealloc).
170     la      t9,myRealloc # Cargo en t9 la direccion de
                        # la myRealloc.
171     jal     ra,t9 # Ejecuto myRealloc con los
                        # parametros: myRealloc(obuffer, amountToSaved*
                        # sizeof(char), (*amountSavedInOBuffer));
172     sw      v0,obuffer # Asigno a obuffer el resultado
                        # de myRealloc almacenado en v0.
173
174     # (obuffer == NULL) ?
175     lw      v0,obuffer
176     bne     v0,DIR_NULL,$CopyLexicoInBuffer # If
                        # (obuffer != NULL) goto CopyLexicoInBuffer
177
178     # obuffer is NULL => Mensaje error.
179     li      a0,FILE_DESCRIPTOR_STDERR
                        # Cargo en a0 FILE_DESCRIPTOR_STDERR.
180     la      a1,MENSAJE_ERROR_MEMORIA_OBUFFER
                        # Cargo en a1 la direccion de memoria
                        # donde se encuentra el mensaje a cargar.

```



```

181      li      a2,
          BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER # Cargo en
          a2 la cantidad de bytes a escribir.
182      li      v0, SYS_write
183      syscall          # No controlo error porque
          sale de por si de la funcion por error.

184
185      li      v0,ERROR_MEMORY
186      sw      v0,56($fp)
187      b      $ReturnProcessDataInIBuffer
188  $CopyLexicoInBuffer:
189      # copyFromLexicoToOBuffer(amountSavedInOBuffer);
190      lw      a0,84($fp) # Cargo en a0
          amountSavedInOBuffer. Parametro de la funcion
          copyFromLexicoToOBuffer.
191      la      t9,copyFromLexicoToOBuffer
192      jal     ra,t9 # Ejecuto la funcion
          copyFromLexicoToOBuffer.

193
194      # int rdoWrite = writeBufferInOFile(
          amountSavedInOBuffer, obuffer);
195      lw      a0,84($fp) # Cargo en a0
          amountSavedInOBuffer. Parametro de la funcion
          writeBufferInOFile.
196      lw      a1,obuffer # Cargo en a1 obuffer. Parametro
          de la funcion writeBufferInOFile.
197      la      t9,writeBufferInOFile
198      jal     ra,t9 # Ejecuto la funcion
          writeBufferInOFile.
199      sw      v0,52($fp) # Guardo en la direccion 52($fp)
          el resultado de writeBufferInOFile.

200
201      # (rdoWrite != OKEY) ?
202      lw      v0,52($fp)
203      beq     v0,OKEY,$WriteInNewOBuffer # If (
          rdoWrite == OKEY) goto WriteInNewOBuffer.

204
205      # rdoWrite is not OKEY
206      lw      v0,52($fp) # Cargo en v0 el codigo de error
          de escribir en el archivo de salida.
207      sw      v0,56($fp) # Guardo en la direccion de
          memoria 56($fp) ese codigo de error.
208      b      $ReturnProcessDataInIBuffer # Salto
          incondicional al return de la funcion.
209  $WriteInNewOBuffer:
210      # *amountSavedInOBuffer = 0;
211      lw      v0,84($fp) # Cargo en v0 lo guardado en la
          direccion de memoria 84($fp), que representaria a
          la variable *amountSavedInOBuffer.
212      sw      zero,0(v0) # Guardo 0 en la direccion
          apuntada por amountSavedInOBuffer.

213
214      # savedInOFile = TRUE;
215      li      v0,TRUE

```

```

216      sw      v0,savedInOFile # Guardo en savedInOFile el
      contenido de v0 (TRUE).
217
218      # obuffer != NULL ?
219      lw      v0,obuffer
220      beq     v0,DIR_NULL,$mymallocNewOBuffer
      # If (obuffer == NULL) goto
      mymallocNewOBuffer.
221
222      # obuffer != NULL => myfree(obuffer) and obuffer =
      NULL.
223      lw      a0,obuffer
224      la      t9,myfree
225      jal     ra,t9          # Ejecuto myfree sobre obuffer.
226      sw      zero,obuffer # Asigno NULL a obuffer.
227 $mymallocNewOBuffer:
228      # obuffer = loadBufferInitial(osize, obuffer);
229      lw      a0,osize      # Cargo en a0 osize. Parametro
      de la funcion loadBufferInitial.
230      lw      a1,obuffer    # Cargo en a1 obuffer.
      Parametro de la funcion loadBufferInitial.
231      la      t9,loadBufferInitial
232      jal     ra,t9          # Ejecuto la funcion
      loadBufferInitial.
233      sw      v0,obuffer    # Asigno la nueva direccion de
      memoria, que se encuentra almacenada en v0, a
      obuffer.
234
235      # (obuffer == NULL) ?
236      lw      v0,obuffer
237      bne     v0,DIR_NULL,$myfreeLexico      # If (
      obuffer != NULL) goto myfreeLexico.
238
239      li      v0,ERROR_MEMORY
240      sw      v0,56($fp)    # Guardo en la direccion de
      memoria 56($fp) ese codigo de error.
241      b       $ReturnProcessDataInIBuffer # Salto
      incondicional al return de la funcion.
242 $LoadLexicoInOBuffer:
243      # (*amountSavedInOBuffer) > 0 ?
244      lw      a0,84($fp)    # Cargo en a0 lo guardado en la
      direccion de memoria 84($fp), que representaria
      a la variable *amountSavedInOBuffer.Parametro de
      la funcion copyFromLexicoToOBuffer.
245      la      t9,copyFromLexicoToOBuffer
246      jal     ra,t9          # Ejecuto la funcion
      copyFromLexicoToOBuffer.
247 $myfreeLexico:
248      lw      a0,lexico     # Cargo en a0 lexico.
249      la      t9,myfree
250      jal     ra,t9          # Ejecuto la funcion myfree
251      sw      zero,lexico   # Asigno NULL a lexico.
252
253      # Dejo quantityCharacterInLexico en 0.

```

```

254         sw         zero,quantityCharacterInLexico
255
256 $VerifyLoadIBuffer:
257     # (idx + 1) == isize ?
258     lw         v0,32($fp) # Cargo en v0 lo guardado en la
                        direccion 32($fp), que seria la variable idx.
259     addu       v1,v0,1    # Incremento en 1 a idx y lo
                        guardo en v1.
260     lw         v0,isize   # Cargo en v0 isize para luego
                        hacer comparacion.
261     bne       v1,v0,$IncrementIdx # If ((idx+1) != isize)
                        goto IncrementIdx
262
263     # ((idx + 1) == isize) is TRUE
264     li         v0,TRUE
265     sw         v0,28($fp) # Guardo en la direccion 28($fp)
                        , que estaba la variable loadIBuffer, TRUE.
266     li         v0,LOAD_I_BUFFER
267     sw         v0,36($fp) # Guardo en la direccion 36($fp)
                        , que estaba la variable rdo -resultado de la
                        operacion-, LOAD_I_BUFFER.
268
269     b         $WhileProcessDataInIBuffer #
                        Salto incondicional al comienzo del while para
                        verificar entrada al mismo.
270 $IncrementIdx:
271     # idx ++
272     lw         v0,32($fp) # Cargo en v0 idx, guardado en
                        la direccion 32($fp).
273     addu       v0,v0,1    # Incremento en 1 a idx.
274     sw         v0,32($fp) # Guardo el nuevo valor de idx.
275
276     b         $WhileProcessDataInIBuffer #
                        Salto incondicional al comienzo del while para
                        verificar entrada al mismo.
277 $LeaveWhileProcessDataInIBuffer:
278     lw         v0,36($fp) # Cargo en v0 el resultado del
                        while: variable rdo guardada en la direccion 36(
                        $fp).
279     sw         v0,56($fp) # Guardo en 56($fp) el resultado
                        de la funcion.
280 $ReturnProcessDataInIBuffer:
281     lw         v0,56($fp)
282     move       sp,$fp
283     lw         ra,72(sp)
284     lw         $fp,68(sp)
285     addu       sp,sp,80
286     j         ra          # Jump and return
287     .end       processDataInIBuffer
288
289
290 ## Mensajes de error
291
292     .rdata

```

```

293         .align 2
294
295 MENSAJE_ERROR_MEMORIA_OBUFFER:
296     .ascii "[Error] Hubo un error de asignacion de
        memoria (obuffer)"
297     .ascii ". \n\000"

```

Stack frame:

Figura 11: Stack frame: processDataInIBuffer

Figura 12: Stack frame: processDataInIBuffer

#### 4.13. Código MIPS32: toLowerCase.S

```

1  #include <mips/regdef.h>
2
3  ##----- toLowerCase -----##
4
5      .align          2
6      .globl          toLowerCase
7      .ent            toLowerCase
8  toLowerCase:
9      .frame          $fp,24,ra
10     .set             noreorder
11     .cpload          t9
12     .set             reorder
13
14     #Stack frame creation
15     subu             sp,sp,24
16
17     .cpstore 0
18     sw               $fp,20(sp)
19     sw               gp,16(sp)
20     move             $fp,sp
21
22     move             v0,a0 # word (this is the character
23     )
24     sb               v0,8($fp)
25     lb               v0,8($fp)
26     slt              v0,v0,65
27     bne              v0,zero,$IfNotLower # if !(word >=
65) goto IfNotLower
28     lb               v0,8($fp)
29     slt              v0,v0,91
30     beq              v0,zero,$IfNotLower # if !(word <=
90) goto IfNotLower
31     lbu              v0,8($fp)
32     addu             v0,v0,32 # word += 32
33     sb               v0,8($fp)

```

```

33 $IfNotLower:
34     lb          v0,8($fp)
35     move        sp,$fp
36
37     #Stack frame destruction.
38     lw          $fp,20(sp)
39     addu        sp,sp,24
40     j           ra # Jump and return
41     .end        toLowerCase

```

Stack frame:

char toLowerCase(char word)			
Offset	Contents	Type reserved area	Comment
24	ra	SRA	
20	fp		
16	gp		
12	a3	ABA	
8	a2    word		Resultado de la función
4	a1		
0	a0		Inicialmente contiene el valor del parametro word.

Figura 13: Stack frame: toLowerCase

#### 4.14. Código MIPS32: verifyPalindromic.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FALSE          0
7  #define TRUE           1
8
9  ##----- verifyPalindromic -----##
10
11     .align             2
12     .globl             verifyPalindromic
13     .ent               verifyPalindromic
14 verifyPalindromic:
15     .frame             $fp,72,ra
16     .set               noreorder
17     .cpload            t9
18     .set               reorder
19
20     #Stack frame creation
21     subu               sp,sp,72
22
23     .cprestore         16
24     sw                 ra,64(sp)
25     sw                 $fp,60(sp)
26     sw                 gp,56(sp)
27     move               $fp,sp

```

```

28
29      sw          a0,72($fp) # char * word
30      sw          a1,76($fp) # int
           quantityCharacterInWord
31
32      lw          v0,72($fp)
33      beq         v0,zero,$IfPalindromicFalse
           # if (word == NULL) goto
           IfPalindromicFalse
34      lw          v0,76($fp)
35      blez        v0,$IfPalindromicFalse
           # if (quantityCharacterInWord <= 0) goto
           IfPalindromicFalse
36      b          $VerifyWhenOneCharacter
           # Salta siempre - goto
           VerifyWhenOneCharacter
37 $IfPalindromicFalse:
38      sw          zero,52($fp) # Guardo FALSE (= 0)
39      b          $ReturnVerifyPalindromic
           # Salta siempre - goto
           ReturnVerifyPalindromic (con return FALSE)
40 $VerifyWhenOneCharacter:
41      lw          v1,76($fp) # Cargo
           quantityCharacterInWord
42      li          v0,1 # Cargo en v0, el valor 1,
           para luego hacer la comparacion
43      bne         v1,v0,$VerifyWhenTwoCaracteres
           # if (quantityCharacterInWord != 1) goto
           VerifyWhenTwoCaracteres
44      li          v0,TRUE # Cargo resultado (
           TRUE es igual a 1)
45      sw          v0,52($fp)
46      b          $ReturnVerifyPalindromic
           # Salta siempre - goto
           ReturnVerifyPalindromic (con return TRUE)
47 $VerifyWhenTwoCaracteres:
48      lw          v1,76($fp) # Cargo
           quantityCharacterInWord
49      li          v0,2 # Cargo en v0, el valor 2,
           para luego hacer la comparacion
50      bne         v1,v0,
           $VerifyWhenMoreThanOneCharacter # if (
           quantityCharacterInWord != 2) goto
           VerifyWhenMoreThanOneCharacter
51
52      # Paso a minuscula el primer caracter del lexico
53      lw          v0,72($fp) # Cargo * word
54      lb          v0,0(v0) # Cargo el primer caracter
           apuntado por word
55      move        a0,v0 # Cargo el primer caracter
           que estaba en v0, en a0. Voy a enviarlo por
           parametro a la funcion toLowerCase
56      la          t9,t0,toLowerCase # Cargo la direccion
           de la funcion toLowerCase

```

```

57     jal      ra,t9      # Salto a la funcion
                    toLowerCase
58     sb       v0,24($fp) # Cargo el resultado en
                    v0 en 24($fp).
59
60     # Paso a minuscula el segundo caracter del lexico
61     lw       v0,72($fp) # Cargo la direccion de
                    memoria en donde esta word
62     addu     v0,v0,1     # Sumo uno a la
                    direccion de memoria, me corro un lugar.
63     lb       v0,0(v0)   # Cargo el segundo
                    caracter apuntado por word (solo habian dos
                    caracteres)
64     move     a0,v0      # Cargo el segundo caracter
                    que estaba en v0, en a0. Voy a enviarlo por
                    parametro a la funcion toLoweCase
65     la       t9,toLowerCase # Cargo la direccion
                    de la funcion toLowerCase
66     jal      ra,t9      # Salto a la funcion
                    toLowerCase
67     sb       v0,25($fp) # Cargo el resultado en
                    v0 en 25($fp).
68
69     lb       v1,24($fp) # Cargo el primer
                    caracter en minuscula en v1
70     lb       v0,25($fp) # Cargo el segundo
                    caracter en minuscula en v0
71     beq      v1,v0,$IfPalindromicTrue
                    # if (firstCharacter == lastCharacter)
                    goto IfPalindromicTrue
72     sw       zero,52($fp) # Guardo FALSE (= 0)
73     b        $ReturnVerifyPalindromic
                    # Salta siempre - goto
                    ReturnVerifyPalindromic (con return FALSE)
74 $IfPalindromicTrue:
75     li       v0,TRUE     # TRUE es igual a 1
76     sw       v0,52($fp)
77     b        $ReturnVerifyPalindromic
                    # Salta siempre - goto
                    ReturnVerifyPalindromic (con return TRUE)
78 $VerifyWhenMoreThanOneCharacter:
79     l.s      $f0,76($fp) # Cargo
                    quantityCharacterInWord
80     cvt.d.w  $f2,$f0     # Convierto el
                    integer quantityCharacterInWord a double
81     l.d      $f0,doubleWord # Cargo en f0 el
                    valor 2.
82     div.d    $f0,$f2,$f0 # Division con Double (
                    double)quantityCharacterInWord / 2; - Sintaxis:
                    div.d FRdest, FRsrc1, FRsrc2
83     s.d      $f0,32($fp) # Guarda el resultado
                    de la division en 32($fp). 0 sea, middle (double
                    middle = (double)quantityCharacterInWord / 2;)

```

```

84      sw          zero,40($fp) # En 40($fp) se
      encuentra idx (int idx = 0;).
85      li          v0,TRUE      # En v0 esta la
      variable validPalindromic en TRUE, que es igual a
      1 (int validPalindromic = TRUE;).
86      sw          v0,44($fp) # Guarda en la direccion
      44($fp) el valor de validPalindromic.
87      lw          v0,76($fp) # Cargo
      quantityCharacterInWord en v0.
88      addu        v0,v0,-1 # Le resto 1 a
      quantityCharacterInWord y lo guardo en v0 (int
      last = quantityCharacterInWord - 1;).
89      sw          v0,48($fp) # Guardo en la direccion
      48($fp) la variable last.
90  $WhileMirror:
91      l.s          $f0,40($fp) # Cargo idx en f0.
92      cvt.d.w      $f2,$f0      # Convierto el
      integer idx a double y lo guardo en f2 para poder
      hacer la comparacion.
93      l.d          $f0,32($fp) # Cargo en a0 la
      variable middle.
94      c.lt.d       $f2,$f0      # Compara la
      variable idx con la variable middle, y setea el
      condition flag en true si el primero (idx) es mas
      chico que el segundo (middle).
95      bc1t        $WhileMirrorConditionLastWithMiddle
      # Si el condition flag es true, continua
      haciendo las comparaciones.
96      b           $WhileMirrorFinalized
      # Si el condition flag es false, salta al
      final de la funcion, devolviendo el valor de la
      variable validPalindromic que seria TRUE.
97  $WhileMirrorConditionLastWithMiddle:
98      l.s          $f0,48($fp) # Cargo la variable
      last en f0.
99      cvt.d.w      $f2,$f0      # Convierto el
      integer last a double y lo guardo en f2 para
      poder hacer la comparacion.
100     l.d          $f0,32($fp) # Cargo en f0 el
      contenido de la variable middle.
101     c.lt.d       $f0,$f2      # Compara el
      contenido de la variable last con la variable
      middle, y setea el condition flag en true si
102     # el primero (last) es mas chico que el segundo (middle)
      .
103     bc1t        $WhileMirrorConditionValidPalindromicTrue # Si el
      condition flag es true, continua haciendo las
      comparaciones.
104     b           $WhileMirrorFinalized
      # Si el condition flag es false, salta al
      final de la funcion, devolviendo el valor de la
      variable validPalindromic que seria TRUE.
105  $WhileMirrorConditionValidPalindromicTrue:

```



```

106      lw          v1,44($fp) # Cargo el contenido de
      la variable validPalindromic, que esta en la
      direccion 44($fp), en v1.
107      li          v0,TRUE    # Cargo TRUE (que
      seria 1) en v0.
108      beq         v1,v0,$WhileMirrorContent
      # If validPalindromic == TRUE goto
      WhileMirrorContent (entro al while).
109      b           $WhileMirrorFinalized
      # Salto para salir del while (bucle).
110 $WhileMirrorContent:
111      # Voy a pasar a minuscula el caracter apuntado desde
      la izquierda.
112      lw          v1,72($fp) # Cargo en v1 el
      contenido de la variable * word.
113      lw          v0,40($fp) # Cargo en v0 el
      contenido de la variable idx.
114      addu        v0,v1,v0 # En v0 coloco el puntero
      a word corrido la cantidad indicada por la
      variable idx.
115      lb          v0,0(v0) # Cargo en v0 el contenido
      de la direccion de memoria (cero corrimiento).
116      move        a0,v0 # Paso a a0 el contenido de
      v0, que seria un unico caracter.
117      la          t9,toLowerCase # Cargo la direccion
      de la funcion toLowerCase
118      jal         ra,t9 # Salto a la funcion
      toLowerCase para pasar el caracter a minuscula.
119      sb          v0,25($fp) # Cargo el caracter
      contenido en v0 a la direccion de memoria 25($fp)
      - char firstCharacter = toLowerCase(word[idx]);
120
121      # Voy a pasar a minuscula el caracter apuntado desde
      la derecha.
122      lw          v1,72($fp) # Cargo en v1 el
      contenido de la variable * word.
123      lw          v0,48($fp) # Cargo en v0 el
      contenido de la variable last.
124      addu        v0,v1,v0 # En v0 coloco el puntero
      a word corrido la cantidad indicada por la
      variable last.
125      lb          v0,0(v0) # Cargo en v0 el contenido
      de la direccion de memoria (cero corrimiento).
126      move        a0,v0 # Paso a a0 el contenido de
      v0, que seria un unico caracter.
127      la          t9,toLowerCase # Cargo la direccion
      de la funcion toLowerCase
128      jal         ra,t9 # Salto a la funcion
      toLowerCase para pasar el caracter a minuscula.
129      sb          v0,24($fp) # Cargo el caracter
      contenido en v0 a la direccion de memoria 24($fp)
      - char lastCharacter = toLowerCase(word[last]);
130

```

```

131      lb          v1,25($fp) # Cargo en v1 el
        contenido de la variable firstCharacter.
132      lb          v0,24($fp) # Cargo en v1 el
        contenido de la variable lastCharacter.
133      beq         v1,v0,$ContinuedInWhileMirror
        # If (firstCharacter == lastCharacter)
        goto ContinuedInWhileMirror
134      sw          zero,44($fp)
135      $ContinuedInWhileMirror:
136      lw          v0,40($fp) # Cargo en v0 el
        contenido de la variable idx.
137      addu        v0,v0,1      # Incremento en uno
        el valor de la variable idx (idx ++).
138      sw          v0,40($fp) # Guardo el contenido de
        la variable idx en la direccion de memoria 40(
        $fp).
139      lw          v0,48($fp) # Cargo en v0 el
        contenido de la variable last.
140      addu        v0,v0,-1    # Decremento en uno el
        valor de la variable last (last --).
141      sw          v0,48($fp) # Guardo el contenido de
        la variable last en la direccion de memoria 48(
        $fp).
142      b           $WhileMirror # Vuelvo a entrar en
        el bucle.
143      $WhileMirrorFinalized:
144      lw          v0,44($fp) # Cargo en v0 el
        contenido de la variable validPalindromic, que se
        encuentra en la direccion de memoria 44($fp).
145      sw          v0,52($fp) # Guardo en la direccion
        de memoria 52($fp) el resultado de la funcion
        verifyPalindromic.
146      $ReturnVerifyPalindromic:
147      lw          v0,52($fp)
148      move        sp,$fp
149      lw          ra,64(sp)
150      lw          $fp,60(sp)
151      addu        sp,sp,72
152      j           ra          # Jump and return
153      .end        verifyPalindromic
154
155
156      ## Variables auxiliares
157
158      .rdata
159      .align      3
160      doubleWord:
161      .word        0
162      .word        1073741824

```

Stack frame:

int verifyPalindromic(char * word, int quantityCharacterInWord)				
Offset	Contents	Type reserved area	Comment	
76	quantityCharacterInWord			
72	* word			
68			nothing to keep	
64	ra			
60	fp	SRA	nothing to keep	
56	gp			
52	Resultado de la función		TRUE    FALSE	
48	last		Inicialmente es igual a quantityCharacterInWord - 1.	
44	validPalindromic		TRUE    FALSE	
40	idx		Inicialmente igual a 0.	
36			nothing to keep	
32	middle	LTA	Es igual a quantityCharacterInWord / 2.	
28			nothing to keep	
25	lastCharacter    firstCharacter			
24	firstCharacter    lastCharacter			
20			nothing to keep	
16			nothing to keep	
12	a3			
8	a2			
4	a1	ABA		Invocación a toLowerCase: 1) un caracter de word -> a0
0	a0			

Figura 14: Stack frame: verifyPalindromic

#### 4.15. Código MIPS32: writeBufferInOFile.S

```

1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #STATICS VAR DEFINITIONS
5
6  #define FALSE                0
7  #define TRUE                 1
8
9  # Resultados de funciones posibles
10 #define OKEY                 0
11 #define ERROR_WRITE          4
12
13
14 ##----- writeBufferInOFile -----##
15
16     .align                    2
17     .globl                    writeBufferInOFile
18     .ent                      writeBufferInOFile
19 writeBufferInOFile:
20     .frame                    $fp,64,ra
21     .set                      noreorder
22     .cpload                   t9
23     .set                      reorder
24
25     #Stack frame creation
26     subu                      sp,sp,64
27
28     .cpstore 16
29     sw                        ra,56(sp)
30     sw                        $fp,52(sp)
31     sw                        gp,48(sp)
32     move                      $fp,sp
33
34     # Parameter

```

```

35      sw          a0,64($fp) # Guardo en la direccion
      de memoria 64($fp) la variable *
      amountSavedInOBuffer (int * amountSavedInOBuffer)
      .
36      sw          a1,68($fp) # Guardo en la direccion
      de memoria 68($fp) la variable * buffer (char *
      buffer).
37
38      sw          zero,24($fp) # Guardo en la
      direccion de memoria 24($fp) la variable
      completeDelivery inicializada
39      # en FALSE (int completeDelivery = FALSE;).
40      sw          zero,28($fp) # Guardo en la
      direccion de memoria 28($fp) la variable
      bytesWriteAcum inicializada
41      # en 0 (int bytesWriteAcum = 0;).
42
43      lw          v0,64($fp) # Cargo en v0 el
      contenido de la direccion de memoria 64($fp), que
      seria la variable * amountSavedInOBuffer.
44      lw          v0,0(v0) # Cargo la direccion de
      memoria del contenido en v0.
45      sw          v0,32($fp) # Guardo en la direccion
      de memoria 32($fp) la direccion de memoria de la
      variable
46      # amountSavedInOBuffer (int bytesToWrite = (*
      amountSavedInOBuffer);).
47      $WhileWriteBufferInOFile:
48      lw          v0,24($fp) # Cargo en v0 el
      contenido de la direccion de memoria 24($fp), que
      seria la variable completeDelivery.
49      beq          v0,FALSE,
      $GoInWhileWriteBufferInOFile # Si
      completeDelivery es FALSE (todavia no se
      guardaron todos los datos cargados en el buffer
      en el archivo)
50      # entro al while para continuar la bajada de los datos
      al buffer.
51      b            $WriteBufferInOFileReturnOkey #
      Salto incondicional para retornar OKEY como
      resultado del proceso de escritura en el archivo
      de salida.
52      $GoInWhileWriteBufferInOFile:
53      # obuffer + bytesWriteAcum
54      lw          v1,68($fp) # Cargo en v1 buffer.
55      lw          v0,28($fp) # Cargo en v0 el
      contenido de la direccion de memoria 28($fp), que
      es la variable bytesWriteAcum.
56      addu        v0,v1,v0 # Sumo la direccion de
      obuffer con el contenido de bytesWriteAcum, y lo
      guardo en v0.
57
58      lw          a0,oFileDescriptor
      # Cargo en a0 la variable oFileDescriptor

```

```

59
60      move          a1,v0    # Muevo el contenido de v0 (
        corrimento de direccion de memoria sobre obuffer
        ) en a1.
61
62      lw            a2,32($fp) # Cargo en a2 el
        contenido de la direccion de memoria 32($fp), que
        seria la variable bytesToWrite.
63
64      li            v0, SYS_write
65      syscall       # Seria write: int bytesWrite = write(
        oFileDescriptor, obuffer + bytesWriteAcum,
        bytesToWrite);
66
67      beq           a3,zero,$SaveBytesWrite
        # Si no hubo error, salto a continuar
        escribiendo
68      # en el archivo si es necesario (
        ContinueWriteBufferInOFile).
69
70      # Hubo un error (la cantidad de caracteres escritos
        es menor a 0, valor negativo).
71      li            v0,ERROR_WRITE # Cargo en v0 el
        resultado de la funcion, que seria un codigo de
        error (ERROR_WRITE).
72      sw            v0,40($fp) # Guardo en la direccion
        de memoria 40($fp) el resultado de la funcion
        que estaba en v0 (ERROR_WRITE).
73      b             $WriteBufferInOFileReturn
        # Salto incondicional al final de la
        funcion, al return.
74 $SaveBytesWrite:
75      # Chequeo errores. v0 contiene el numero de
        caracteres escrito (es negativo si hubo error).
76      sw            v0,36($fp) # Guardo en la direccion
        de memoria 36($fp) la cantidad de bytes escritos
77      # efectivamente en el archivo de salida, que esta en v0.
78 $ContinueWriteBufferInOFile:
79      # bytesWriteAcum += bytesWrite;
80      lw            v1,28($fp) # Cargo en v1 el
        contenido de la direccion de memoria 28($fp), que
        seria la variable bytesWriteAcum.
81      lw            v0,36($fp) # Cargo en v0 el
        contenido de la direccion de memoria 36($fp), que
        seria la variable bytesWrite.
82      addu          v0,v1,v0 # Sumo el contenido de v1
        (bytesWriteAcum) y el contenido de v0 (bytesWrite
        ), y guardo el resultado en v0.
83      sw            v0,28($fp) # Guardo en la direccion
        de memoria 28($fp) el contenido de v0, que seria
        el resultado
84      # de la suma (bytesWriteAcum += bytesWrite;).
85

```

```

86      # bytesToWrite = (*amountSavedInOBuffer) -
      bytesWriteAcum;
87      lw      v0,64($fp) # Cargo en v0 el
      contenido de la direccion de memoria 64($fp), que
      seria la variable
88      # *amountSavedInOBuffer (una direccion de memoria).
89      lw      v1,0(v0) # Cargo lo contenido en la
      direccion de memoria guardada en v0 en v1 (
      ammountSavedInOBuffer es un puntero).
90      lw      v0,28($fp) # Cargo en v0 el
      contenido de la direccion de memoria 28($fp), que
      seria la variable bytesWriteAcum.
91      subu    v0,v1,v0 # Resto el contenido de v1
      (*amountSavedInOBuffer) con el contenido de v0 (
      bytesWriteAcum), y
92      # guardo el resultado en v0.
93      sw      v0,32($fp) # Guardo el resultado de
      la resta en la direccion de memoria 32($fp), que
      seria la variable bytesToWrite.
94
95      lw      v0,32($fp) # Cargo en v0 el
      contenido de la direccion de memoria 32($fp), que
      seria la variable bytesToWrite.
96      bgtz    v0,$WhileWriteBufferInOFile
      # Si bytesToWrite es mayor a cero, salto
      a WhileWriteBufferInOFile (vuelvo a entrar al
      loop while).
97      # (bytesToWrite <= 0) ? then:
98      li      v0,TRUE # Cargo en v0 el
      literal TRUE (que es 1).
99      sw      v0,24($fp) # Guardo en la direccion
      de memoria 24($fp) el contenido de v0. 0 sea,
      completeDelivery = TRUE;
100     b        $WhileWriteBufferInOFile
      # Salto incondicional al inicio del loop.
101 $WriteBufferInOFileReturnOkey:
102     sw      zero,40($fp) # Guardo en la
      direccion de memoria 40($fp) el resultado OKEY (
      resultado de la funcion writeBufferInOFile).
103 $WriteBufferInOFileReturn:
104     lw      v0,40($fp) # Cargo el resultado de
      la funcion writeBufferInOFile, que estaba en la
      direccion de
105     # memoria 40($fp), en el registro v0.
106     move    sp,$fp
107     lw      ra,56(sp)
108     lw      $fp,52(sp)
109     addu    sp,sp,64
110     j        ra # Jump and return
111     .end    writeBufferInOFile

```

Stack frame:

int writeBufferInOFfile(int * amountSavedInBuffer, char * buffer)				
Offset	Contents	Type reserved area	Comment	
68	* amountSavedInOBuffer			
64	* buffer			
60			nothing to keep	
56	ra	SRA		
52	fp			
48	gp			
44			nothing to keep	
40	Resultado de la función	LTA	OKEY    Error	
36	bytesWrite			
32	bytesToWrite		Inicialmente es igual a * amountSavedInOBuffer.	
28	bytesWriteAcum		Inicialmente es igual a 0.	
24	completeDelivery		FALSE    TRUE	
20			nothing to keep	
16		ABA	nothing to keep	
12	a3		Cada vez que se invoca a SYS_write (para informar errores), se guarda en a3 si hubo o no error. a3 = 0 => no hubo error.	Invocación a SYS_write: 1) ofileDescriptor -> a0    dirección sobre obuffer -> a1    bytesToWrite -> a2
8	a2			
4	a1		Inicialmente contiene el valor del parametro * buffer.	
0	a0		Inicialmente contiene el valor del parametro * amountSavedInOBuffer.	

Figura 15: Stack frame: writeBufferInOFfile

## 5. Ejecución

A continuación algunos de los comandos válidos para la ejecución del programa:

Comandos usando un archivo de entrada y otro de salida

```
$ tp1 -i input.txt -o output.txt
```

```
$ tp1 --input input.txt --output output.txt
```

Comando para la salida standard

```
$ tp1 -i input.txt
```

Comando para el ingreso standard

```
$ tp1 -o output.txt
```

Por defecto los tamaños del buffer in y buffer out son 1 byte. puede especificar el tamaño a usar los mismos en la llamada.

```
$ tp1 -i input.txt -o output.txt -I 10 -O 10
```

-I: indica el tamaño (bytes) a usar por el buffer in

-O: indica el tamaño (bytes) a usar por el buffer out

### 5.1. Comandos para ejecución

Desde el netBSD ejecutar:

Para compilar el código

```
$ gcc -Wall -o tp1 tp1.c process.S
```

-Wall: activa los mensajes de warnning

-o: indica el archivo de salida.

Para obtener el código MIPS32 del proyecto c:

```
$ gcc -Wall -O0 -S -mrnames tp1.c
```

-S: detiene el compilador luego de generar el código assembly  
-mrnames: indica al compilador que genere la salida con nombre de registros  
-O0: indica al compilador que no aplique optimizaciones.

## 5.2. Análisis sobre tiempo de ejecución

Comando para la medición del tiempo (time):

```
$ time ./tp1 -i ../input-large.txt -I 10 -O 10
```

Se midieron y se tuvieron en cuenta los tiempo transcurridos entre distintas ejecuciones cambiando los parámetros de entrada de buffer in y buffer out. Para medir se usó la instrucción "time" la cual arroja los tiempos efectivamente consumidos por el CPU en la ejecución del programa. A continuación una tabla con los valores medidos:

Tamaño de archivo usado aproximadamente 1,70 MB.  
Tamaño de línea en archivo aproximadamente: 1 byte \* 450 char = 450 byte(caracteres/línea).

id	stream input	stream output	real time[s]	user time[s]	sys time[s]
1	1	1	82,53	9,74	36,68
2	2	2	50,55	8,16	21,11
3	3	2	30,70	7,78	8,72
4	5	5	19,34	6,94	4,67
5	10	10	11,42	7,10	1,10
6	50	50	8,90	6,85	0,66
7	100	100	7,97	6,97	0,34
8	300	300	7,45	6,76	0,22
9	600	600	7,53	6,90	0,18
10	1000	1000	7,58	6,89	0,25
11	1500	1500	7,33	6,84	0,24
12	2000	2000	7,40	6,86	0,23
13	2500	2500	7,43	6,94	0,11
14	3500	3500	7,18	6,70	0,16

Cuadro 1: Valores de exe medidos(time).

Cómo puede verse en la figura las ejecuciones iniciales con valores bajos de lectura y escritura(buffer 1 byte) tienen tiempos de respuesta del programa elevados; mientras que a medida que se aumenta el tamaño del buffer los tiempos van creciendo hasta un limite asintótico alrededor de 7 segundos.





Figura 16: Gráfico de incidencia del buffer

Es de notar que un pequeño aumento en el tamaño del buffer(in/out) aumenta la performance considerablemente.

### 5.3. Comandos para ejecución de tests

Comando para ejecutar el test automático

```
$ bash test-automatic.sh
```

La salida debería ser la siguiente(todos los test OK):

```
#####
##### Tests automaticos
#####
#####
###-----### COMIENZA test ejercicio 1 del informe.
###-----###
###-----### STDIN ::: FILE OUTPUT
###-----###
OK
###-----### FIN test ejercicio 1 del informe.
###-----###
###-----###
###-----###
###-----###
###-----### COMIENZA test ejercicio 2 del informe.
###-----###
###-----### FILE INPUT ::: STDOUT
###-----###
OK
###-----### FIN test ejercicio 2 del informe.
###-----###
###-----###
###-----###
###-----### COMIENZA test con -i - -o -
###-----###
###-----### STDIN ::: STDOUT
###-----###
OK
###-----### FIN test con -i - -o -
###-----###
###-----###
###-----###
###-----### COMIENZA test palabras con acentos
###-----###
OK
###-----### FIN test palabras con acentos
###-----###
###-----###
###-----###
```

```

###-----### COMIENZA test con caritas
###-----###
OK
###-----### FIN test con caritas
###-----###
###-----###
###-----###
###-----###
###-----### COMIENZA test con entrada estandar
###-----###
OK
###-----### FIN test con entrada estandar
###-----###
###-----###
###-----###
###-----###
###-----### COMIENZA test con salida estandar
###-----###
OK
###-----### FIN test con salida estandar
###-----###
###-----###
###-----###
###-----###
###-----### COMIENZA test con entrada y salida estanda
###-----###
OK
###-----### FIN test con entrada y salida estanda
###-----###
###-----###
###-----###
###-----###
###-----### COMIENZA test menu version (-V)
###-----###
OK
###-----### FIN test menu version (-V)
###-----###
###-----###
###-----###
###-----###
###-----### COMIENZA test menu version (--version)
###-----###
OK
###-----### FIN test menu version (--version)
###-----###
###-----###
###-----###
###-----###
###-----### COMIENZA test menu help (-h)
###-----###
OK
###-----### FIN test test menu help (-h)
###-----###
###-----###

```

```

#####
###-----### COMIENZA test menu help (--help)
      ###-----###
OK
###-----### FIN test menu help (--help)
      ###-----###
#####
#####
#####
##### Tests automaticos
      #####
#####
#####
#-----# COMIENZA test con /-o -i - #-----#
OK

```

## 6. Stack frame de funciones

A continuación los stack frame de las diferentes funciones:

## 7. Conclusiones

A través del presente trabajo se logro realizar una implementación pequeña de un programa c y assembly MIPS32. La invocación desde un programa assembly a un programa c; la implementación de una función malloc, free y realloc en código assembly, sin hacer uso de la implementación c. La forma de llamar a funciones de

Por otro lado se logró familiarizarse con la implementación de assembly MIPS y con la ABI.

La implementación de la función palindroma con un buffer permitió ver que en función de la cantidad de caracteres leídos cada vez, el tiempo de ejecución del programa disminuía considerablemente. Al mismo tiempo la mejora en el tiempo de ejecución tiene un límite a partir del cual un aumento en el tamaño del buffer no garantiza ganancia en la ejecución del programa.

## Referencias

- [1] Intel Technology & Research, “Hyper-Threading Technology,” 2006, <http://www.intel.com/technology/hyperthread/>.
- [2] J. L. Hennessy and D. A. Patterson, “Computer Architecture. A Quantitative Approach,” 3ra Edición, Morgan Kaufmann Publishers, 2000.
- [3] J. Larus and T. Ball, “Rewriting Executable Files to Mesure Program Behavior,” Tech. Report 1083, Univ. of Wisconsin, 1992.  
<https://es.wikipedia.org/wiki/Pal>

int isKeywords(char character)			
Offset	Contents	Type reserved area	Comment
24	ra	SRA	Save a return address
20	fp		Save a frame pointer
16	gp		Save global pointer
12	Resultado de la función palindrome isKeywords.	ABA	Result of palindrome isKeywords
8	character		Character
4	-		<b>nothing to keep</b>
0	-		<b>nothing to keep</b>

Figura 17: Stack frame 1

char toLowerCase(char word)			
Offset	Contents	Type reserved area	Comment
24	ra	SRA	Save return address
20	fp		Save a frame pointer
16	gp		Save global pointer
12	1   0	ABA	Result of function: True = 1; False = 0
8	character		Save a word param
4	-		<b>nothing to keep</b>
0	-		<b>nothing to keep</b>

Figura 18: Stack frame 2

void * myRealloc(void * ptr, size_t tamanyoNew, int tamanyoOld)			
Offset	Contents	Type registers area	Comment
48	ra	SRA	save a return address
44	fp		save a frame pointer
40	a0		original allocation size
36	a0   t1		actual allocation size   pointer
32	v0   t0		save name function > alloc size
28	0		<b>nothing to keep</b>
24	0		<b>nothing to keep</b>
20	t0		-
16	t0	ABA	-
12	-		without alloc
8	-		without alloc
4	t1		point to new memory
0	-		without alloc

Figura 19: Stack frame 3