Trabajo Práctico Nro. 1: programación MIPS: Reentrega

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 palindroma. La función estará escrita en assembly MIPS.

Los streams serán leídos y escritos de a bloques de memoría 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.
- Setting del tamaño del buffer in y buffer out; indicando de a cuántos caracteres se debe leer y escribir.

A continuación un gráfico que muestra la disposición de la implementación:

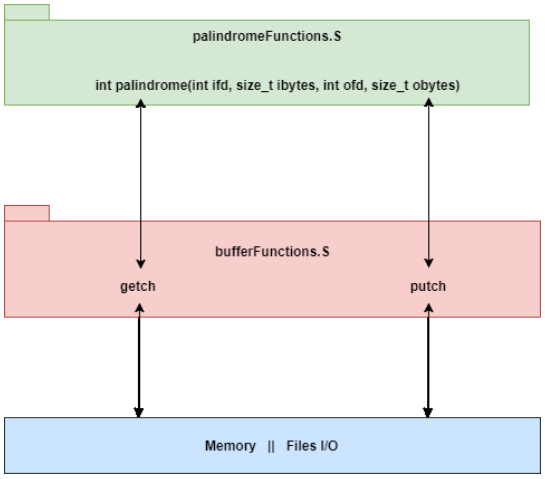

Figura 1: Se tiene dos grandes paquetes de funciones que hacen al proyecto, por un lado las asociadas a la funcionalidad de verificacion de léxicos palíndromos; y por otro lado, el encargado de proveer los caracteres que pueden ser parte de un léxico.

3. Implementación

3.1. Código fuente en lenguaje C: tp1.c

```c
/*
 ============================================================================

 Name        : tp1.c
 Author      : Grupo orga 66.20
 Version     : 1
 Copyright   : Orga6620 - Tp1
 Description : Trabajo practico 1: Programacion MIPS
 ============================================================================

 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <unistd.h>

#include "constants.h"
#include "palindromeFunctions.h"

#define VERSION "1.2"

size_t isize = 1;
size_t osize = 1;

int executeHelp() {
        fprintf(stdout, "Usage: \n");
        fprintf(stdout, "       tp1 -h \n");
        fprintf(stdout, "       tp1 -V \n");
        fprintf(stdout, "       tp1 [options] \n");
        fprintf(stdout, "Options: \n");
        fprintf(stdout, "       -V, --version           Print
            version and quit. \n");
        fprintf(stdout, "       -h, --help              Print this
            information. \n");
        fprintf(stdout, "       -i, --input             Location of
            the input file. \n");
        fprintf(stdout, "       -o, --output            Location of
            the output file. \n");
        fprintf(stdout, "       -I, --ibuf-bytes        Byte-count
            of the input buffer. \n");
        fprintf(stdout, "       -O, --obuf-bytes        Byte-count
            of the output buffer. \n");
        fprintf(stdout, "Examples: \n");
        fprintf(stdout, "       tp1 -i ~/input -o ~/output \n");

        return OKEY;
}

int executeVersion() {
        fprintf(stdout, "Version: \"%s\" \n", VERSION);

        return OKEY;
}

int executeByMenu(int argc, char **argv) {
        int inputFileDefault = FALSE;
        int outputFileDefault = FALSE;
        FILE * fileInput = stdin;
        FILE * fileOutput = stdout;

        // Always begins with /
        if (argc == 1) {
                // Run with default parameters
                inputFileDefault = TRUE;
                outputFileDefault = TRUE;
```

```
60          }
61
62          char * pathInput = NULL;
63          char * pathOutput = NULL;
64          char * iBufBytes = NULL;
65          char * oBufBytes = NULL;
66
67          /* Una cadena que lista las opciones cortas validas */
68          const char* const smallOptions = "Vhi:o:I:O:";
69
70          /* Una estructura de varios arrays describiendo los valores
                largos */
71          const struct option longOptions[] = {
72                  {"version",             no_argument,            0,
                        'V' },
73                  {"help",                no_argument,            0,
                        'h' },
74                  {"input",               required_argument,  0,  'i'
                        }, // optional_argument
75                  {"output",              required_argument,  0,  'o'
                        },
76                  {"ibuf-bytes",  required_argument,      0,  'I' },
77                  {"obuf-bytes",  required_argument,      0,  'O' },
78                  {0,                     0,                      0,
                            0   }
79          };
80
81          int incorrectOption = FALSE;
82          int finish = FALSE;
83          int result = OKEY;
84          int longIndex = 0;
85          char opt = 0;
86          /*
87           * Switch para obtener los parámetros de entrada.
88           */
89          while ((opt = getopt_long(argc, argv, smallOptions,
90                                    longOptions, &longIndex )) != -1
                                        && incorrectOption == FALSE
                                        && finish == FALSE) {
91                  switch (opt) {
92                          case 'V' :
93                                  result = executeVersion();
94                                  finish = TRUE;
95                                  break;
96                          case 'h' :
97                                  result = executeHelp();
98                                  finish = TRUE;
99                                  break;
100                         case 'i' :
101                                 pathInput = optarg;
102                                 break;
103                         case 'o' :
104                                 pathOutput = optarg;
105                                 break;
106                         case 'I' :
107                                 iBufBytes = optarg;
108                                 break;
109                         case 'O' :
110                                 oBufBytes = optarg;
111                                 break;
112                         default:
113                                 incorrectOption = TRUE;
114                 }
115         }
116
117         if (incorrectOption == TRUE) {
118                 fprintf(stderr, "[Error] Incorrecta option de menu.\
                        n");
119                 return INCORRECT_MENU;
120         }
121
122         if (finish == TRUE) {
123                 return result;
124         }
```

```
125
126         if (iBufBytes != NULL) {
127                 char *finalPtr;
128                 isize = strtoul(iBufBytes, &finalPtr, 10);
129                 if (isize == 0) {
130                         fprintf(stderr, "[Error] Incorrecta cantidad
                                de bytes para el buffer de entrada.\n")
                               ;
131                         return ERROR_BYTES;
132                 }
133         }
134
135         if (oBufBytes != NULL) {
136                 char *finalPtr;
137                 osize = strtoul(oBufBytes, &finalPtr, 10);
138                 if (osize == 0) {
139                         fprintf(stderr, "[Error] Incorrecta cantidad
                                de bytes para el buffer de salida.\n");
140                         return ERROR_BYTES;
141                 }
142         }
143
144         if (pathInput == NULL || strcmp("-",pathInput) == 0) {
145                 inputFileDefault = TRUE;
146         }
147
148         if (pathOutput == NULL || strcmp("-",pathOutput) == 0) {
149                 outputFileDefault = TRUE;
150         }
151         /*
152          * Se abren los ficheros de lectura y escritura.
153          * Se chequea si hubo errores en la apertura.
154          */
155         if (inputFileDefault == FALSE) {
156                 fileInput = fopen(pathInput, "r"); // Opens an
                        existing text file for reading purpose.
157                 if (fileInput == NULL) {
158                         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          * Obtenemos el file descriptor number.
180          */
181         int ifd = fileno(fileInput);
182         int ofd = fileno(fileOutput);
183         /*
184          * Llamado a función principal
185          */
```

```
186         int executeResult = palindrome(ifd, isize, ofd, osize);
187
188         int resultFileInputClose = 0; // EOF = -1
189
190         /*
191          * Se cierran los ficheros de lectura y escritura.
192          * Se cheqeuea si hubo errores en la cierre.
193          */
194         if (inputFileDefault == FALSE && fileInput != NULL) {
195                 resultFileInputClose = fclose(fileInput);
196                 if (resultFileInputClose == EOF) {
197                         fprintf(stderr, "[Warning] El archivo de
                                input no pudo ser cerrado correctamente:
                                 %s \n", pathInput);
198                 }
199         }
200
201         if (outputFileDefault == FALSE && fileOutput != NULL) {
202                 int result = fclose(fileOutput);
203                 if (result == EOF) {
204                         fprintf(stderr, "[Warning] El archivo de
                                output no pudo ser cerrado correctamente
                                : %s \n", pathOutput);
205                         resultFileInputClose = EOF;
206                 }
207         }
208
209         if (resultFileInputClose != 0) {
210                 return ERROR_FILE;
211         }
212
213         return executeResult;
214 }
215
216 /*
217  * Chequeo cantidad de parámetros.
218  * Ejecución de menú.
219  */
220 int main(int argc, char **argv) {
221         // / -i lalala.txt -o pepe.txt -I 2 -O 3 => 9 parameters as
                maximum
222         if (argc > 9) {
223                 fprintf(stderr, "[Error] Cantidad máxima de pará
                        metros incorrecta: %d \n", argc);
224                 return INCORRECT_QUANTITY_PARAMS;
225         }
226
227         return executeByMenu(argc, argv);
228 }
```

3.2. Código fuente en lenguaje C: bufferFunctions.c

```c
/*
 * bufferFunctions.c
 *
 */

#include "bufferFunctions.h"

/*** input  ***/
int ifd = 0;
int lastPositionInIBufferRead = -1;
Buffer ibuffer = { NULL, 0, 0 };
//Determina si el input file tiene un EOF
int endIFile = FALSE;

/*** output  ***/
int ofd = 0;
Buffer obuffer = { NULL, 0, 0 };


void initializeInput(int iFileDescriptor, size_t ibytes) {
        ifd = iFileDescriptor;
        ibuffer.sizeBytes = ibytes;
}

void initializeOutput(int oFileDescriptor, size_t obytes) {
        ofd = oFileDescriptor;
        obuffer.sizeBytes = obytes;
}
/*
 * Carga en el input buffer con caracteres.
 */
int loadIBufferWithIFile() {

        /*
         * Reservo memoria para alocar caracteres leídos.
         * La determinación del buffer se encuentra en el parámetro
         * de entrada en la llamada al programa.
         */
        if (ibuffer.buffer == NULL) {
                ibuffer.buffer = (char *) malloc(ibuffer.sizeBytes*
                    sizeof(char));
                if (ibuffer.buffer == NULL) {
                        fprintf(stderr, "[Error] Hubo un error de
                            asignacion de memoria (ibuffer). \n");
                        return ERROR_MEMORY;
                }
        }

        int completeDelivery = FALSE;
        ibuffer.quantityCharactersInBuffer = 0;
        int bytesToRead = ibuffer.sizeBytes;

        // Lleno el buffer de entrada
        while (completeDelivery == FALSE && endIFile == FALSE) {
                int bytesRead = read(ifd, ibuffer.buffer + ibuffer.
                    quantityCharactersInBuffer, bytesToRead);
                if (bytesRead == -1) {
                        fprintf(stderr, "[Error] Hubo un error en la
                            lectura de datos del archivo. \n");
                        return ERROR_I_READ;
                }

                if (bytesRead == 0) {
                        endIFile = TRUE;
                }

                ibuffer.quantityCharactersInBuffer += bytesRead;
                bytesToRead = ibuffer.sizeBytes - ibuffer.
                    quantityCharactersInBuffer;

```

```
66                  if (bytesToRead <= 0) {
67                          completeDelivery = TRUE;
68                  }
69          }
70
71          lastPositionInIBufferRead = -1;
72
73          return OKEY_I_FILE;
74  }
75
76  /*
77   * Obtengo un caracter(char) del input file o del buffer.
78   * Lo seteo en el buffer.
79   */
80  int getch() {
81          if (ibuffer.buffer == NULL || lastPositionInIBufferRead == (
                ibuffer.quantityCharactersInBuffer - 1)) {
82                  if (endIFile == TRUE) {
83                          return EOF;
84                  }
85                  int resultLoadIBuffer = loadIBufferWithIFile();
86                  if (resultLoadIBuffer == ERROR_I_READ) {
87                          return ERROR_I_READ;
88                  }
89
90                  if (ibuffer.quantityCharactersInBuffer == 0) {
91                          return EOF;
92                  }
93          }
94
95          lastPositionInIBufferRead ++;
96          return ibuffer.buffer[lastPositionInIBufferRead];
97  }
98
99  /*
100  * Escribe los caracteres en el output file
101  * de acuerdo al tamaño del buffer.
102  */
103 int writeBufferInOFile() {
104         if (obuffer.buffer == NULL || obuffer.
                quantityCharactersInBuffer <= 0) {
105                 return OKEY;
106         }
107
108         int completeDelivery = FALSE;
109         int bytesWriteAcum = 0;
110         int bytesToWrite = obuffer.quantityCharactersInBuffer;
111         while (completeDelivery == FALSE) {
112                 int bytesWrite = write(ofd, obuffer.buffer +
                        bytesWriteAcum, bytesToWrite);
113                 if (bytesWrite < 0) {
114                         fprintf(stderr, "[Error] Hubo un error al
                                escribir en el archivo. \n");
115                         return ERROR_WRITE;
116                 }
117
118                 bytesWriteAcum += bytesWrite;
119                 bytesToWrite = obuffer.quantityCharactersInBuffer -
                        bytesWriteAcum;
120
121                 if (bytesToWrite <= 0) {
122                         completeDelivery = TRUE;
123                 }
124         }
125
126         return OKEY;
127 }
128
129 /*
130  * Coloca un char en el output buffer.
131  * Llama a la escritura en el output file
132  * de ser necesario.
133  */
134 int putch(int character) {
```

```
135         if (obuffer.buffer == NULL) {
136                 obuffer.buffer = (char *) malloc(obuffer.sizeBytes*
                        sizeof(char));
137                 if (obuffer.buffer == NULL) {
138                         fprintf(stderr, "[Error] Hubo un error de
                                asignacion de memoria (obuffer). \n");
139                         return ERROR_MEMORY;
140                 }
141
142                 obuffer.quantityCharactersInBuffer = 0;
143         }
144
145         obuffer.buffer[obuffer.quantityCharactersInBuffer] =
                character;
146         obuffer.quantityCharactersInBuffer ++;
147
148         if (obuffer.quantityCharactersInBuffer == obuffer.sizeBytes)
                {
149                 writeBufferInOFile();
150                 obuffer.quantityCharactersInBuffer = 0;
151         }
152
153         return OKEY;
154 }
155 /*
156  * Flusea el contenido del buffer.
157  * Guarda el contenido en el archivo de salida.
158  */
159 int flush() {
160         if (obuffer.buffer != NULL && obuffer.
                quantityCharactersInBuffer > 0) {
161                 return writeBufferInOFile();
162         }
163
164         return OKEY;
165 }
166
167 /*
168  * Libera los recursos solicitados por ibuffer/obuffer.
169  */
170 void freeResources() {
171         if (ibuffer.buffer != NULL) {
172                 free(ibuffer.buffer);
173                 ibuffer.buffer = NULL;
174         }
175
176         if (obuffer.buffer != NULL) {
177                 free(obuffer.buffer);
178                 obuffer.buffer = NULL;
179         }
180 }
181
182 /*
183  * Carga el caracter en el buffer
184  */
185 int loadInBuffer(char character, Buffer * buffer, size_t sizeInitial
        ) {
186         if (buffer->buffer == NULL) {
187                 buffer->buffer = malloc(sizeInitial * sizeof(char));
188                 buffer->sizeBytes = sizeInitial;
189         } else if (buffer->quantityCharactersInBuffer >= buffer->
                sizeBytes) {
190                 size_t bytesLexicoPreview = buffer->sizeBytes;
191                 //Se hace una reasignacion exponencial del espacio.
192                 buffer->sizeBytes = bytesLexicoPreview * 2;
193                 // Esto es para no perder memoria.
194                 char * auxiliary = myRealloc(buffer->buffer, buffer
                        ->sizeBytes*sizeof(char), bytesLexicoPreview);
195                 if (auxiliary == NULL) {
196                         cleanContentBuffer(buffer);
197                 } else {
198                         buffer->buffer = auxiliary;
199                 }
200         }
```

```
201
202         if (buffer->buffer == NULL) {
203                 fprintf(stderr, "[Error] Hubo un error en memoria (
                        buffer). \n");
204                 return ERROR_MEMORY;
205         }
206
207         buffer->buffer[buffer->quantityCharactersInBuffer] =
                character;
208         buffer->quantityCharactersInBuffer ++;
209
210         return OKEY;
211 }
212
213 /*
214  * Limpia el contenido del buffer pasado por parámetro.
215  */
216 void cleanContentBuffer(Buffer * buffer) {
217         if (buffer->buffer != NULL) {
218                 free(buffer->buffer);
219                 buffer->buffer = NULL;
220         }
221
222         buffer->quantityCharactersInBuffer = 0;
223         buffer->sizeBytes = 0;
224 }
```

3.3. Código fuente en lenguaje C: memoryFunctions.c

```
1  /*
2   * memoryFunctions.c
3   *
4   */
5  #include "memoryFunctions.h"
6
7  void * myRealloc(void * ptr, size_t tamanyoNew, int tamanyoOld) {
8          if (tamanyoNew <= 0) {
9                  free(ptr);
10                 ptr = NULL;
11
12                 return NULL;
13         }
14
15         void * ptrNew = (void *) malloc(tamanyoNew);
16         if (ptrNew == NULL) {
17                 return NULL;
18         }
19
20         if (ptr == NULL) {
21                 return ptrNew;
22         }
23
24         int end = tamanyoNew;
25         if (tamanyoOld < tamanyoNew) {
26                 end = tamanyoOld;
27         }
28
29         char *tmp = ptrNew;
30         const char *src   = ptr;
31
32         while (end--) {
33                 *tmp = *src;
34                 tmp++;
35                 src++;
36         }
37
38         free(ptr);
39         ptr = NULL;
40
41         return ptrNew;
42 }
```

3.4. Código fuente en lenguaje C: palindromeFunctions.c

```c
/*
 * palindromeFunctions.c
 *
 */

#include "palindromeFunctions.h"

/*
 * Contiene la palabra leida.
 */
Buffer lexico;

/*
 * Pasa los caracteres válidos de mayúscula a minúscula.
 */
char toLowerCase(char word) {

        if (word >= 65 && word <= 90) {
                word += 32;
        }

        return word;
}

/*
 * Pre: Léxico siempre contiene caracteres válidos.
 * Verifica que el lexico(palabra) sea palindroma.
 */
int verifyPalindromic() {
        if (lexico.buffer == NULL || lexico.
            quantityCharactersInBuffer <= 0) {
                return FALSE;
        }

        /*
         * Las palabras de 1 sólo caracter(válido)
         * son siempre palindromas.
         */
        if (lexico.quantityCharactersInBuffer == 1) {
                // The word has one character
                return TRUE;
        }

        double middle = (double)lexico.quantityCharactersInBuffer /
            2;
        int idx = 0;
        int validPalindromic = TRUE;
        int last = lexico.quantityCharactersInBuffer - 1;
        while(idx < middle && last >= middle && validPalindromic ==
            TRUE) {
                char firstCharacter = toLowerCase(lexico.buffer[idx
                    ]);
                char lastCharacter = toLowerCase(lexico.buffer[last
                    ]);
                if (firstCharacter != lastCharacter) {
                        validPalindromic = FALSE;
                }

                idx ++;
                last --;
        }

        return validPalindromic;
}

/*
 * Verifica si un determinado caracter es un
 * 'caracter válido' para procesar.
 */
int isKeywords(char character) {
```

```
66          /* ASCII:
67           *              A - Z = [65 - 90]
68           *              a - z = [97 - 122]
69           *              0 - 9 = [48 - 57]
70           *              - =  45
71           *              _ = 95
72           */
73          if ((character >= 65 && character <= 90) || (character >= 97
                 && character <= 122)
74                          || (character >= 48 && character <= 57)
75                          || character == 45 || character == 95) {
76                  return TRUE;
77          }
78
79          return FALSE;
80  }
81
82  /*
83   * Verifica si es palindromo.
84   * Si es palindromo, llama a putch para enviar el char
85   * al buffer.
86   * Si es palindromo, realiza putch de todos los char
87   * que contiene el léxico.
88   */
89  int saveIfPalindrome() {
90          int itsPalindromic = verifyPalindromic();
91
92          if (itsPalindromic == TRUE) {
93                  int idx = 0;
94                  int error = FALSE;
95                  while(idx < lexico.quantityCharactersInBuffer &&
                         error == FALSE) {
96                          int result = putch(lexico.buffer[idx]);
97                          if (result == EOF) {
98                                  error = TRUE;
99                          }
100                         idx ++;
101                 }
102
103                 if (error == FALSE) {
104                         int result = putch('\n');
105                         if (result == EOF) {
106                                 error = TRUE;
107                         }
108                 }
109
110                 if (error == TRUE) {
111                         fprintf(stderr, "[Error] Error al escribir
                                 en el archivo output el palindromos. \n"
                                 );
112                         return ERROR_PUTCH;
113                 }
114         }
115         return OKEY;
116 }
117
118 /*
119  * Función principal.
120  * Si el palindromo es válido lo carga en el buffer.
121  */
122 int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes) {
123         initializeInput(ifd, ibytes);
124         initializeOutput(ofd, obytes);
125
126         lexico.quantityCharactersInBuffer = 0;
127         int icharacter = getch();
128         int result = OKEY;
129         while (icharacter != EOF && icharacter != ERROR_I_READ &&
                 result == OKEY) {
130                 char character = icharacter;
131
132                 if (isKeywords(character) == TRUE) {
133                         result = loadInBuffer(character, &lexico,
                                 LEXICO_BUFFER_SIZE);
```

```
134                     } else {
135                             // Dentro de esta funcion se invoca a putch
                                    si el lexico es palindromo.
136                             result = saveIfPalindrome();
137
138                             cleanContentBuffer(&lexico);
139                     }
140
141                     icharacter = getch();
142             }
143
144             // Guardo lo que haya quedado en lexico si es palindromo.
145             int resultFlush = saveIfPalindrome();
146             if (result == OKEY) {
147                     result = resultFlush;
148             }
149
150             cleanContentBuffer(&lexico);
151
152             resultFlush = flush();
153             if (result == OKEY) {
154                     result = resultFlush;
155             }
156             freeResources();
157
158
159             return result;
160     }
```

4. Código MIPS32

4.1. Código MIPS32: bufferFunctions.S

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #include "constants.h"
5  ##include "memoryFunctions.h"
6
7  # Size mensajes
8  #define BYTES_MENSAJE_ERROR_MEMORIA_IBUFFER      60
9  #define BYTES_MENSAJE_ERROR_LECTURA_ARCHIVO      60
10 #define BYTES_MENSAJE_ERROR_ESCRITURA_ARCHIVO    51
11 #define BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER      60
12 #define BYTES_MENSAJE_ERROR_MEMORIA_BUFFER       59
13
14 ##------ initializeInput ------##
15
16         .text
17         .align  2
18         .globl  initializeInput
19         .ent    initializeInput
20 initializeInput:
21         .frame  $fp,16,ra
22         .set    noreorder
23         .cpload t9
24         .set    reorder
25
26         # Stack frame creation
27         subu    sp,sp,16
28
29         .cprestore 0
30         sw      $fp,12(sp)
31         sw      gp,8(sp)
32
33         # de aqui al fin de la funcion uso $fp en lugar de sp.
34         move    $fp,sp
35
36         # Parametros
37         sw      a0,16($fp)      # Guardo en la direccion de memoria
               16($fp) la
38                                 # variable iFileDescriptor (int).
39         sw      a1,20($fp)      # Guardo en la direccion de memoria
               20($fp) la
40                                 # variable ibytes (size_t).
41
42         # ifd = iFileDescriptor;
43         lw      v0,16($fp)      # Cargo en v0 iFileDescriptor.
44         sw      v0,ifd          # Guardo el contenido de v0,
               iFileDescriptor, en
45                                 # la variable ifd.
46
47         # ibuffer.sizeBytes = ibytes;
48         lw      v0,20($fp)      # Cargo en v0 ibytes.
49         sw      v0,ibuffer+8    # Guardo en sizeBytes (ibuffer+8) el
               contenido
50                                 # de v0 (ibytes).
51
52         move    sp,$fp
53         lw      $fp,12(sp)
54         # destruyo stack frame
55         addu    sp,sp,16
56         # vuelvo a funcion llamante
57         j       ra
58
59         .end    initializeInput
60
61
62
63 ##------ initializeOutput ------##
```

```
64
65          .align  2
66          .globl  initializeOutput
67          .ent    initializeOutput
68  initializeOutput:
69          .frame  $fp,16,ra
70          .set    noreorder
71          .cpload t9
72          .set    reorder
73
74          # Stack frame creation
75          subu    sp,sp,16
76
77          .cprestore 0
78          sw      $fp,12(sp)
79          sw      gp,8(sp)
80
81          # de aqui al fin de la funcion uso $fp en lugar de sp.
82          move    $fp,sp
83
84          # Parametros
85          sw      a0,16($fp)      # Guardo en la direccion de memoria
                16($fp) la
86                                  # variable oFileDescriptor (int).
87          sw      a1,20($fp)      # Guardo en la direccion de memoria
                20($fp) la
88                                  # variable obytes (size_t).
89
90          # ofd = oFileDescriptor;
91          lw      v0,16($fp)      # Cargo en v0 oFileDescriptor.
92          sw      v0,ofd          # Guardo el contenido de v0,
                oFileDescriptor, en
93                                  # la variable ofd.
94
95          # obuffer.sizeBytes = obytes;
96          lw      v0,20($fp)      # Cargo en v0 obytes.
97          sw      v0,obuffer+8    # Guardo en sizeBytes (obuffer+8) el
                 contenido
98                                  # de v0 (obytes).
99
100         move    sp,$fp
101         lw      $fp,12(sp)
102         # destruyo stack frame
103         addu    sp,sp,16
104         # vuelvo a funcion llamante
105         j       ra
106
107         .end    initializeOutput
108
109
110
111 ##------ loadIBufferWithIFile ------##
112
113         .align  2
114         .globl  loadIBufferWithIFile
115         .ent    loadIBufferWithIFile
116 loadIBufferWithIFile:
117         .frame  $fp,56,ra
118         .set    noreorder
119         .cpload t9
120         .set    reorder
121
122         # Stack frame creation
123         subu    sp,sp,56
124
125         .cprestore 16
126         sw      ra,48(sp)
127         sw      $fp,44(sp)
128         sw      gp,40(sp)
129
130         # de aqui al fin de la funcion uso $fp en lugar de sp.
131         move    $fp,sp
132
133         # (ibuffer.buffer == NULL)
```

```
134         lw      v0,ibuffer      # Cargo en v0 ibuffer. El primer
                campo del struct
135                                 # es buffer.
136         bne     v0,zero,ibufferNotNull # If (ibuffer.buffer != NULL)
                 goto ibufferNotNull
137
138         # ibuffer.buffer is NULL
139         lw      a0,ibuffer+8    # Cargo en a0 la variable sizeBytes
                (ibuffer.sizeBytes).
140                                 # Es el tercer elemento del struct
                    Buffer.
141                                 # Es parametro de la funcion
                    mymalloc.
142         la      t9,mymalloc     # Cargo en t9 la direccion de
                memoria de mymalloc.
143         jal     ra,t9           # Ejecuto la funcion mymalloc.
144
145         # Verifico asignacion de memoria.
146         sw      v0,ibuffer      # Asigno la memoria reservada con
                mymalloc a ibuffer.buffer
147         lw      v0,ibuffer      # Cargo en v0 ibuffer.buffer
148         bne     v0,zero,ibufferNotNull # If (ibuffer.buffer != NULL)
                 goto ibufferNotNull
149
150         # ibuffer.buffer is NULL  => Mensaje de error.
151         li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
152         la      a1,MENSAJE_ERROR_MEMORIA_IBUFFER # Cargo en a1 la
                direccion de memoria donde se encuentra el mensaje a
                cargar.
153         li      a2,BYTES_MENSAJE_ERROR_MEMORIA_IBUFFER # Cargo en a2
                 la cantidad de bytes a escribir.
154         li      v0, SYS_write
155         syscall   # No controlo error porque sale de por si de la
                funcion por error.
156         li      v0,ERROR_MEMORY
157         sw      v0,36($fp)      # Guardo en la direccion de memoria
                36($fp) el codigo de error.
158                                 # Es el resultado de la funcion
                    loadIBufferWithFile.
159         b       returnLoadIBufferWithFile # Salto incodicional al
                return de la funcion.
160 ibufferNotNull:
161         # ibuffer.buffer is not NULL
162
163         # int completeDelivery = FALSE;
164         sw      zero,24($fp)    # Guardo en la direccion de memoria
                24($fp) la variable
165                                 # completeDelivery, inicializada en
                    FALSE (= 0).
166
167         # ibuffer.quantityCharactersInBuffer = 0;
168         sw      zero,ibuffer+4  # Asigno 0 a la variable ibuffer.
                quantityCharactersInBuffer.
169                                 # quantityCharactersInBuffer es el
                    segundo elemento del struct
                    Buffer.
170
171         # int bytesToRead = ibuffer.sizeBytes;
172         lw      v0,ibuffer+8    # Cargo en v0 el valor de sizeBytes.
173                                 # sizeBytes es el tercer elemento
                    del struct Buffer (Buffer
                    ibuffer).
174         sw      v0,28($fp)      # Guardo en la direccion 28($fp) el
                valor de sizeBytes, que representaria
175                                 # a la variable bytesToRead.
176
177         # Lleno el buffer de entrada
178 whileLoadIBuffer:
179         # (completeDelivery == FALSE && endIFile == FALSE)
180
181         # (completeDelivery == FALSE)
182         lw      v0,24($fp)      # Cargo en v0 la variable
                completeDelivery, guardada
```

```
183                                 # en la direccion 24($fp).
184         bne     v0,FALSE,initializeLastPositionInIBufferRead # If (
                completeDelivery != FALSE)
185         # goto initializeLastPositionInIBufferRead
186
187         # completeDelivery is FALSE
188
189         # (endIFile == FALSE)
190         lw      v0,endIFile     # Cargo en v0 endIFile.
191         bne     v0,FALSE,initializeLastPositionInIBufferRead # If (
                endIFile != FALSE)
192         # goto initializeLastPositionInIBufferRead
193
194         # completeDelivery is FALSE && endIFile is FALSE
195
196         # int bytesRead = read(ifd, ibuffer.buffer + ibuffer.
                quantityCharactersInBuffer, bytesToRead);
197         lw      v1,ibuffer      # Cargo en v1 ibuffer.buffer (primer
                 elemento del struct Buffer).
198         lw      v0,ibuffer+4    # Cargo en v0 ibuffer.
                quantityCharactersInBuffer (segundo
199                                 # elemento del struct Buffer).
200         addu    v0,v1,v0        # Me muevo sobre ibuffer.buffer.
                Guarda esta direccion de memoria
201                                 # en v0.
202
203         lw      a0,ifd          # Cargo en a0 ifd, parametro de la
                funcion read.
204         move    a1,v0           # Cargo la direccion de memoria que
                estaba en v0 en a1. Parametro
205                                 # de la funcion read.
206         lw      a2,28($fp)      # Cargo en a2 bytesToRead que estaba
                 en la direccion 28($fp).
207         li      v0, SYS_read
208         syscall     # Seria read: int bytesRead = read(ifd, ibuffer.
                buffer +
209                     #               ibuffer.
                            quantityCharactersInBuffer, bytesToRead);
210
211         # Controlo errores y cantidad de bytes leidos. v0 contiene
                el numero de caracteres
212         # leidos (es negativo si hubo error y es 0 si llego a fin
                del archivo).
213
214         beq     a3,zero,savedBytesRead #Si a3 es cero, no hubo error
215
216         # Hubo error en la lectura de los datos => Mensaje de error.
217
218         li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
219         la      a1,MENSAJE_ERROR_LECTURA_ARCHIVO # Cargo en a1 la
                direccion de memoria donde
220                                                  # se encuentra el
                                                         mensaje a cargar
                                                         .
221         li      a2,BYTES_MENSAJE_ERROR_LECTURA_ARCHIVO # Cargo en a2
                la cantidad de bytes a escribir.
222         li      v0, SYS_write
223         syscall  # No controlo error porque sale de por si de la
                funcion por error.
224
225         li      v0,ERROR_I_READ
226         sw      v0,36($fp)      # Guardo en la direccion de memoria
                36($fp) el codigo de error.
227                                 # Es el resultado de la funcion
                                        loadIBufferWithFile.
228         b       returnLoadIBufferWithFile # Salto incodicional al
                return de la funcion.
229
230 savedBytesRead:
231         sw      v0,32($fp)      # Guardo en la direccion de memoria
                40($fd) el resultado de la
232                                 # funcion read, que estaria
                                        representado por la variable
```

```
                                        bytesRead.
233
234         # (bytesRead == 0)
235         lw      v0,32($fp)      # Cargo en v0 lo que esta en la
                direccion 32($fp), que seria bytesRead.
236         bne     v0,zero,loadBytesRead # If (bytesRead != 0) goto
                loadBytesRead.
237
238         # bytesRead is 0
239
240         # endIFile = TRUE;
241         li      v0,TRUE
242         sw      v0,endIFile
243 loadBytesRead:
244         # ibuffer.quantityCharactersInBuffer += bytesRead;
245         lw      v1,ibuffer+4    # Cargo en v1 ibuffer.
                quantityCharactersInBuffer (segundo
246                                 # elemento del struct Buffer).
247         lw      v0,32($fp)      # Cargo en v0 los bytes leidos (
                bytesRead).
248         addu    v0,v1,v0        # Guardo el resultado de la suma en
                v0:
249                                 # quantityCharactersInBuffer +
                                        bytesRead.
250         sw      v0,ibuffer+4    # Guardo el resultado de la suma en
                ibuffer.quantityCharactersInBuffer.
251
252         # bytesToRead = ibuffer.sizeBytes - ibuffer.
                quantityCharactersInBuffer;
253         lw      v1,ibuffer+8    # Cargo en v1 ibuffer.sizeBytes.
254         lw      v0,ibuffer+4    # Cargo en v0 ibuffer.
                quantityCharactersInBuffer.
255         subu    v0,v1,v0        # Guardo el resultado de la suma en
                v0:
256                                 # ibuffer.sizeBytes - ibuffer.
                                        quantityCharactersInBuffer
257         sw      v0,28($fp)      # Guardo el resultado de la suma en
                la direccion 28($fp), que
258                                 # representa la variable bytesToRead
                                        .
259
260         # (bytesToRead <= 0)
261         lw      v0,28($fp)      # Cargo en v0 bytesToRead.
262         bgtz    v0,whileLoadIBuffer # If (bytesToRead > 0) goto
                whileLoadIBuffer
263
264         # bytesToRead is <= 0
265
266         # completeDelivery = TRUE;
267         li      v0,TRUE
268         sw      v0,24($fp)      # Guardo TRUE en la direccion 24($fp
                ), que
269                                 # representa la variable
                                        completeDelivery.
270         b       whileLoadIBuffer # Vuelvo a intentar entrar al loop.
271 initializeLastPositionInIBufferRead:
272         # lastPositionInIBufferRead = -1;
273         li      v0,-1
274         sw      v0,lastPositionInIBufferRead
275
276         # return OKEY_I_FILE;
277         li      v0,OKEY_I_FILE
278         sw      v0,36($fp)
279 returnLoadIBufferWithFile:
280         lw      v0,36($fp)
281         move    sp,$fp
282         lw      ra,48(sp)
283         lw      $fp,44(sp)
284         # destruyo stack frame
285         addu    sp,sp,56
286         # vuelvo a funcion llamante
287         j       ra
288
289         .end    loadIBufferWithIFile
```

```
290
291
292
293 ##------ getch ------##
294
295         .align  2
296         .globl  getch
297         .ent    getch
298 getch:
299         .frame  $fp,48,ra
300         .set    noreorder
301         .cpload t9
302         .set    reorder
303
304         # Stack frame creation
305         subu    sp,sp,48
306
307         .cprestore 16
308         sw      ra,40(sp)
309         sw      $fp,36(sp)
310         sw      gp,32(sp)
311
312         # de aqui al fin de la funcion uso $fp en lugar de sp.
313         move    $fp,sp
314
315         # (ibuffer.buffer == NULL || lastPositionInIBufferRead == (
                ibuffer.quantityCharactersInBuffer - 1))
316         lw      v0,ibuffer      # Cargo en v0 ibuffer.buffer (primer
                 elemento del struct Buffer).
317         beq     v0,zero,verifyEndIFile  # If (ibuffer.buffer == NULL
                ) goto verifyEndIFile
318
319         # ibuffer.buffer is not NULL
320
321         # (lastPositionInIBufferRead == (ibuffer.
                quantityCharactersInBuffer - 1))
322         lw      v0,ibuffer+4    # Cargo en v0 ibuffer.
                quantityCharactersInBuffer (segundo
323                                 # elemento del struct Buffer).
324         addu    v1,v0,-1        # Cargo en v1 el resultadod de la
                suma:
325                                 # ibuffer.quantityCharactersInBuffer
                                        + (-1)
326         lw      v0,lastPositionInIBufferRead # Cargo en v0
                lastPositionInIBufferRead
327         beq     v0,v1,verifyEndIFile # If (lastPositionInIBufferRead
                 == (ibuffer.quantityCharactersInBuffer - 1))
328                                 # goto verifyEndIFile
329
330         b       increaseLastPositionInIBufferRead # Salto
                incondicional.
331 verifyEndIFile:
332         # (endIFile == TRUE)
333         lw      v1,endIFile     # Cargo en v1 endIFile.
334         li      v0,TRUE
335         bne     v1,v0,loadIBuffer # If (endIFile != TRUE) goto
                loadIBuffer
336
337         # endIFile is TRUE
338         li      v0,EOF_F        # EOF_F = -1
339         sw      v0,28($fp)      # Guardo en la direccion 28($fp) el
                resultado de la funcion.
340         b       returnGetch     # Salto incondicional al return de
                la funcion.
341 loadIBuffer:
342         # int resultLoadIBuffer = loadIBufferWithIFile();
343         la      t9,loadIBufferWithIFile
344         jal     ra,t9           # Ejecuto la funcion
                loadIBufferWithIFile
345         sw      v0,24($fp)      # Guardo el resultado de la funcion
                en la direccion 24($fp),
346                                 # que representaria la variable
                                        resultLoadIBuffer.
347
```

```
348         # (resultLoadIBuffer == ERROR_I_READ)
349         lw      v1,24($fp)      # Cargo en v1 la variable de
                resultLoadIBuffer.
350         li      v0,ERROR_I_READ
351         bne     v1,v0,verifyQuantityCharactersInBuffer # If (
                resultLoadIBuffer != ERROR_I_READ)
352         # goto verifyQuantityCharactersInBuffer
353
354         # return ERROR_I_READ;
355         li      v0,ERROR_I_READ
356         sw      v0,28($fp)
357         b       returnGetch
358 verifyQuantityCharactersInBuffer:
359         # (ibuffer.quantityCharactersInBuffer == 0)
360         lw      v0,ibuffer+4    # Cargo en v0 ibuffer.
                quantityCharactersInBuffer (segundo
361                                 # elemento del struct Buffer).
362         # If (ibuffer.quantityCharactersInBuffer != 0) goto
                increaseLastPositionInIBufferRead
363         bne     v0,zero,increaseLastPositionInIBufferRead
364
365         # return EOF_F;
366         li      v0,EOF_F        # EOF_F = -1
367         sw      v0,28($fp)
368         b       returnGetch
369 increaseLastPositionInIBufferRead:
370         # lastPositionInIBufferRead ++;
371         lw      v0,lastPositionInIBufferRead
372         addu    v0,v0,1
373         sw      v0,lastPositionInIBufferRead
374
375         # return ibuffer.buffer[lastPositionInIBufferRead];
376         lw      v1,ibuffer
377         lw      v0,lastPositionInIBufferRead
378         addu    v0,v1,v0
379         lb      v0,0(v0)
380         sw      v0,28($fp)
381 returnGetch:
382         lw      v0,28($fp)
383         move    sp,$fp
384         lw      ra,40(sp)
385         lw      $fp,36(sp)
386         # destruyo stack frame
387         addu    sp,sp,48
388         # vuelvo a funcion llamante
389         j       ra
390
391        .end     getch
392
393
394
395 ###------ writeBufferInOFile ------##
396
397         .align  2
398         .globl  writeBufferInOFile
399         .ent    writeBufferInOFile
400 writeBufferInOFile:
401         .frame  $fp,64,ra
402         .set    noreorder
403         .cpload t9
404         .set    reorder
405
406         # Stack frame creation
407         subu    sp,sp,64
408
409         .cprestore 16
410         sw      ra,56(sp)
411         sw      $fp,52(sp)
412         sw      gp,48(sp)
413
414         # de aqui al fin de la funcion uso $fp en lugar de sp.
415         move    $fp,sp
416
417         # (obuffer.buffer == NULL || obuffer.
```

```
418             quantityCharactersInBuffer <= 0)
419
420         # (obuffer.buffer == NULL)
421         lw      v0,obuffer      # Cargo en v0 obuffer.buffer (primer
                elemento del
422                                 # struct Buffer.
423         beq     v0,zero,returnOkey # If (obuffer.buffer == NULL)
                goto returnOkey
424
425         # obuffer.buffer is not NULL
426
427         # (obuffer.quantityCharactersInBuffer <= 0)
428         lw      v0,obuffer+4    # Cargo en v0 obuffer.
                quantityCharactersInBuffer
429                                 # (segundo elemento del struct
                                        Buffer).
430         blez    v0,returnOkey   # If (obuffer.
                quantityCharactersInBuffer <= 0)
431                                 # goto returnOkey.
432         b       loadOBuffer     # Salto incondicional a loadOBuffer.
433 returnOkey:
434         # return OKEY;
435         sw      zero,40($fp)    # Guardo en la direccion 40($fp) el
                resultado de la
436                                 # funcion, en este caso OKEY (= 0).
437         b       returnWriteBufferInOFile # Salto incondicional al
                return de la funcion.
438 loadOBuffer:
439         # int completeDelivery = FALSE (= 0)
440         sw      zero,24($fp)    # 24($fp) <-> completeDelivery
441
442         # int bytesWriteAcum = 0;
443         sw      zero,28($fp)    # 28($fp) <-> bytesWriteAcum
444
445         # int bytesToWrite = obuffer.quantityCharactersInBuffer;
446         lw      v0,obuffer+4    # quantityCharactersInBuffer es el
                segundo elemento del
447                                 # struct Buffer.
448         sw      v0,32($fp)      # 32($fp) <-> bytesToWrite
449 whileWriteOFile:
450         # (completeDelivery == FALSE)
451         lw      v0,24($fp)      # Cargo en v0 completeDelivery
452         beq     v0,FALSE,inWhileWriteOFile  # If (completeDelivery
                == FALSE) goto inWhileWriteOFile
453
454         # completeDelivery is not FALSE (is TRUE)
455         b       loadReturnOkey # Salto incondicional a
                loadReturnOkey.
456 inWhileWriteOFile:
457         # int bytesWrite = write(ofd, obuffer.buffer +
                bytesWriteAcum, bytesToWrite);
458
459         # obuffer.buffer + bytesWriteAcum
460         lw      v1,obuffer      # obuffer.buffer es el primer
                elemento del struct Buffer.
461         lw      v0,28($fp)      # Cargo en v0 bytesWriteAcum.
462         addu    v0,v1,v0        # Me muevo por buffer, guardo la
                direccion en v0.
463
464         lw      a0,ofd          # Cargo en a0 ofd. Parametro de la
                funcion write.
465         move    a1,v0           # Cargo en a1 la direccion sobre
                obuffer.buffer.
466                                 # Parametro de la funcion write.
467         lw      a2,32($fp)      # Cargo en a2 bytesToWrite. Parametro
                 de la funcion write.
468
469         li      v0, SYS_write
470         syscall  # Seria int bytesWrite = write(ofd, obuffer.buffer
                + bytesWriteAcum, bytesToWrite);
471
472         beq     a3,zero,saveBytesWrite # Si no hubo error, salto a
                saveBytesWrite.
```

```
473         # Hubo error al querer escribir en el archivo => Mensaje de
                error.
474         li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
475         la      a1,MENSAJE_ERROR_ESCRITURA_ARCHIVO # Cargo en a1 la
                direccion de memoria donde se encuentra el mensaje a
                cargar.
476         li      a2,BYTES_MENSAJE_ERROR_ESCRITURA_ARCHIVO # Cargo en
                a2 la cantidad de bytes a escribir.
477         li      v0, SYS_write
478         syscall # No controlo error porque sale de por si de la
                funcion por error.
479
480         # return ERROR_WRITE;
481         li      v0,ERROR_WRITE  # Cargo codigo de error, que sera el
                 resultado de la funcion.
482         sw      v0,40($fp)      # Guardo en la direccion 40($fp) el
                resultado de la funcion.
483         b       returnWriteBufferInOFile
484 saveBytesWrite:
485         sw      v0,36($fp)      # Guardo en la direccion 36($fp) los
                bytes escritor,
486                                 # que representarian la variable
                                        bytesWrite.
487         # bytesWriteAcum += bytesWrite;
488         lw      v1,28($fp)      # 28($fp) <-> bytesWriteAcum. Cargo
                en v1 bytesWriteAcum
489         lw      v0,36($fp)      # Cargo en v0 bytesWrite.
490         addu    v0,v1,v0        # Sumo estos dos valores y guardo
                resultado en v0.
491         sw      v0,28($fp)      # Guardo en bytesWriteAcum su nuevo
                valor (resultado de la suma).
492
493         # bytesToWrite = obuffer.quantityCharactersInBuffer -
                bytesWriteAcum;
494         lw      v1,obuffer+4    # obuffer.quantityCharactersInBuffer
                 es el segundo elemento
495                                 # del struct Buffer. Cargo este
                                        valor en v1.
496         lw      v0,28($fp)      # Cargo en v0 bytesWriteAcum.
497         subu    v0,v1,v0        # Resto estos dos valores. Guardo
                resultado en v0.
498         sw      v0,32($fp)      # Asigno a bytesToWrite el resultado
                 de la resta.
499
500         # (bytesToWrite <= 0)
501         lw      v0,32($fp)      # Cargo en v0 bytesToWrite
502         bgtz    v0,whileWriteOFile # If (bytesToWrite > 0) goto
                whileWriteOFile
503
504         # bytesToWrite is <= 0
505         li      v0,TRUE
506         sw      v0,24($fp)      # Asigno a completeDelivery TRUE
507         b       whileWriteOFile
508 loadReturnOkey:
509         sw      zero,40($fp)    # OKEY = 0
510 returnWriteBufferInOFile:
511         lw      v0,40($fp)
512         move    sp,$fp
513         lw      ra,56(sp)
514         lw      $fp,52(sp)
515         # destruyo stack frame
516         addu    sp,sp,64
517         # vuelvo a funcion llamante
518         j       ra
519
520         .end    writeBufferInOFile
521
522
523
524 ###------ putch ------##
525
526         .align  2
527         .globl  putch
```

```
528         .ent    putch
529 putch:
530         .frame  $fp,48,ra
531         .set    noreorder
532         .cpload t9
533         .set    reorder
534 
535         # Stack frame creation
536         subu    sp,sp,48
537 
538        .cprestore 16
539         sw      ra,40(sp)
540         sw      $fp,36(sp)
541         sw      gp,32(sp)
542 
543         # de aqui al fin de la funcion uso $fp en lugar de sp.
544         move    $fp,sp
545 
546         # Parametro
547         sw      a0,48($fp)      # Guardo en la direccion de memoria
                48($fp) lo que tiene a0, que
548                                 # es el parametro que recibe la
                                        funcion, int character.
549 
550         # (obuffer.buffer == NULL)
551         lw      v0,obuffer      # Cargo en v0 obuffer.buffer, que es
                 el primer elemento del
552                                 # struct Buffer.
553         bne     v0,zero,loadInOBuffer # If (obuffer.buffer != NULL)
                goto loadInOBuffer
554 
555         # obuffer.buffer is NULL
556 
557         # Asigno memoria a obuffer.buffer
558         # obuffer.buffer = (char *) malloc(obuffer.sizeBytes*sizeof(
                char));
559         lw      a0,obuffer+8    # quantityCharactersInBuffer es el
                tercer elemento del struct Buffer.
560         la      t9,mymalloc
561         jal     ra,t9
562         sw      v0,obuffer
563         lw      v0,obuffer
564 
565         # Verifico error en la asignacion de memoria.
566         # (obuffer.buffer == NULL)
567         bne     v0,zero,initializeQuantityCharactersInOBuffer
568         # If (obuffer.buffer != NULL) goto
569         # initializeQuantityCharactersInOBuffer.
570 
571         # obuffer.buffer is NULL => Mensaje de error.
572         li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
573         la      a1,MENSAJE_ERROR_MEMORIA_OBUFFER # Cargo en a1 la
                direccion de memoria donde se encuentra el mensaje a
                cargar.
574         li      a2,BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER  # Cargo en
                a2 la cantidad de bytes a escribir.
575         li      v0, SYS_write
576         syscall     # No controlo error porque sale de por si de la
                funcion por error.
577 
578         # return ERROR_MEMORY;
579         li      v0,ERROR_MEMORY
580         sw      v0,24($fp)      # Cargo en la direccion 24($fp) el
                resultado de la funcion,
581                                 # en este caso es el codigo de error
                                        ERROR_MEMORY.
582         b       returnPutch     # Salto incondicional a returnPutch.
583 initializeQuantityCharactersInOBuffer:
584         # obuffer.quantityCharactersInBuffer = 0;
585         sw      zero,obuffer+4
586 loadInOBuffer:
587         # obuffer.buffer[obuffer.quantityCharactersInBuffer] =
                character;
```

```
588         lw      v1,obuffer      # Cargo obuffer.buffer en v1.
589         lw      v0,obuffer+4    # Cargo obuffer.
                quantityCharactersInBuffer en v0.
590         addu    v1,v1,v0        # Me muevo sobre obuffer.buffer, o
                sea:
591                                 # obuffer.buffer[obuffer.
                                        quantityCharactersInBuffer]
592                                 # Guardo esta direccion en v1.
593         lbu     v0,48($fp)      # Cargo en v0 character.
594         sb      v0,0(v1)        # Asigno character a obuffer.buffer[
                obuffer.quantityCharactersInBuffer].
595
596         # obuffer.quantityCharactersInBuffer ++;
597         lw      v0,obuffer+4
598         addu    v0,v0,1
599         sw      v0,obuffer+4
600
601         # (obuffer.quantityCharactersInBuffer == obuffer.sizeBytes)
602         lw      v1,obuffer+4    # obuffer.quantityCharactersInBuffer
                es el segundo elemento del
603                                 # struct Buffer.
604         lw      v0,obuffer+8    # obuffer.sizeBytes es el tercer
                elemento del struct Buffer.
605         bne     v1,v0,loadReturnPutch # If (obuffer.
                quantityCharactersInBuffer != obuffer.sizeBytes) goto
                loadReturnPutch
606
607         # obuffer.quantityCharactersInBuffer is equal obuffer.
                sizeBytes
608
609         # writeBufferInOFile();
610         la      t9,writeBufferInOFile
611         jal     ra,t9
612
613         # obuffer.quantityCharactersInBuffer = 0;
614         sw      zero,obuffer+4
615 loadReturnPutch:
616         # return OKEY;
617         sw      zero,24($fp)
618 returnPutch:
619         lw      v0,24($fp)
620         move    sp,$fp
621         lw      ra,40(sp)
622         lw      $fp,36(sp)
623         # destruyo stack frame
624         addu    sp,sp,48
625         # vuelvo a funcion llamante
626         j       ra
627
628         .end    putch
629
630
631
632 ##------ flush ------##
633
634         .align  2
635         .globl  flush
636         .ent    flush
637 flush:
638         .frame  $fp,48,ra
639         .set    noreorder
640         .cpload t9
641         .set    reorder
642
643         # Stack frame creation
644         subu    sp,sp,48
645
646         .cprestore 16
647         sw      ra,40(sp)
648         sw      $fp,36(sp)
649         sw      gp,32(sp)
650
651         # de aqui al fin de la funcion uso $fp en lugar de sp.
```

```
652         move    $fp,sp
653
654         # (obuffer.buffer != NULL && obuffer.
                quantityCharactersInBuffer > 0)
655
656         # (obuffer.buffer != NULL)
657         lw      v0,obuffer
658         beq     v0,zero,loadReturnOkeyFlush # If (obuffer.buffer ==
                NULL) goto loadReturnOkeyFlush.
659
660         # obuffer.buffer is equal NULL
661
662         # (obuffer.quantityCharactersInBuffer > 0)
663         lw      v0,obuffer+4    # obuffer.quantityCharactersInBuffer
                es el segundo
664                                 # elemento del struct Buffer.
665         blez    v0,loadReturnOkeyFlush # If (obuffer.
                quantityCharactersInBuffer <= 0)
666                                   # goto loadReturnOkeyFlush
667
668         # obuffer.quantityCharactersInBuffer is > 0
669
670         # return writeBufferInOFile();
671         la      t9,writeBufferInOFile
672         jal     ra,t9
673         sw      v0,24($fp)    # Cargo en la direccion 24($fp) el
                resultado de ejecutar
674                               # la funcion writeBufferInOFile.
675         b       returnFlush
676 loadReturnOkeyFlush:
677         # return OKEY;
678         sw      zero,24($fp)  # OKEY = 0
679 returnFlush:
680         lw      v0,24($fp)
681         move    sp,$fp
682         lw      ra,40(sp)
683         lw      $fp,36(sp)
684         # destruyo stack frame
685         addu    sp,sp,48
686         # vuelvo a funcion llamante
687         j       ra
688
689         .end    flush
690
691
692
693 ##------ freeResources ------##
694
695         .align  2
696         .globl  freeResources
697         .ent    freeResources
698 freeResources:
699         .frame  $fp,40,ra
700         .set    noreorder
701         .cpload t9
702         .set    reorder
703
704         # Stack frame creation
705         subu    sp,sp,40
706
707         .cprestore 16
708         sw      ra,32(sp)
709         sw      $fp,28(sp)
710         sw      gp,24(sp)
711
712         # de aqui al fin de la funcion uso $fp en lugar de sp.
713         move    $fp,sp
714
715         # (ibuffer.buffer != NULL)
716         lw      v0,ibuffer
717         beq     v0,zero,freeOBufferBuffer # If (ibuffer.buffer ==
                NULL) goto freeOBufferBuffer
718
719         # ibuffer.buffer is not NULL
```

```
720
721         # free(ibuffer.buffer);
722         lw      a0,ibuffer      # ibuffer.buffer es el primer elemento
                del struct Buffer
723                                 # (Buffer ibuffer).
724         la      t9,myfree
725         jal     ra,t9
726
727         # ibuffer.buffer = NULL;
728         sw      zero,ibuffer
729 freeOBufferBuffer:
730         # (obuffer.buffer != NULL)
731         lw      v0,obuffer
732         beq     v0,zero,returnFreeResources # If (obuffer.buffer ==
                NULL) goto returnFreeResources
733
734         # obuffer.buffer is not NULL
735
736         # free(obuffer.buffer);
737         lw      a0,obuffer      # obuffer.buffer es el primer elemento
                del struct Buffer
738                                 # (Buffer ibuffer).
739         la      t9,myfree
740         jal     ra,t9
741
742         # obuffer.buffer = NULL;
743         sw      zero,obuffer
744 returnFreeResources:
745         move    sp,$fp
746         lw      ra,32(sp)
747         lw      $fp,28(sp)
748         # destruyo stack frame
749         addu    sp,sp,40
750         # vuelvo a funcion llamante
751         j       ra
752
753         .end    freeResources
754
755
756
757 ##------ loadInBuffer ------##
758
759         .align  2
760         .globl  loadInBuffer
761         .ent    loadInBuffer
762 loadInBuffer:
763         .frame  $fp,56,ra
764         .set    noreorder
765         .cpload t9
766         .set    reorder
767
768         # Stack frame creation
769         subu    sp,sp,56
770
771         .cprestore 16
772         sw      ra,52(sp)
773         sw      $fp,48(sp)
774         sw      gp,44(sp)
775         sw      s0,40(sp)
776
777         # de aqui al fin de la funcion uso $fp en lugar de sp.
778         move    $fp,sp
779
780         move    v0,a0           # Cargo en v0 lo que viene en a0,
                que es character (char).
781         sw      a1,60($fp)      # Cargo en la direccion 60($fp) lo
                que viene en a1, que
782                                 # es * buffer (Buffer * buffer).
783         sw      a2,64($fp)      # Cargo en 64($fp) lo que viene en
                a2, que es sizeInitial (size_t).
784         sb      v0,24($fp)      # Guardo en la direccion 24($fp) lo
                que estaba en v0, que era
785                                 # el parametro character.
786
```

```
787         # (buffer->buffer == NULL)
788         lw      v0,60($fp)      # Cargo en v0 la direccion de buffer
                .
789         lw      v0,0(v0)        # Cargo en v0 el contenido en esa
                direccion de memoria, que
790                                 # que seria buffer->buffer.
791         bne     v0,zero,compareQuantities # If (buffer->buffer !=
                NULL) goto compareQuantities.
792
793         # buffer->buffer is NULL
794
795         # buffer->buffer = malloc(sizeInitial * sizeof(char));
796         lw      s0,60($fp)      # Cargo en s0 la direccion de buffer
                .
797         lw      a0,64($fp)      # Cargo en a0 sizeInitial, parametro
                 para mymalloc.
798         la      t9,mymalloc
799         jal     ra,t9           # Ejecuto mymalloc.
800         sw      v0,0(s0)        # Asigno la memoria a lo que apunta
                buffer (buffer -> buffer).
801                                 # La validacion de NULL en la
                                        asignacion de memoria se hace
                                        luego.
802
803         # buffer->sizeBytes = sizeInitial;
804         lw      v1,60($fp)      # Cargo en v1 la direccion de buffer
                .
805         lw      v0,64($fp)      # Cargo en v0 sizeInitial.
806         sw      v0,8(v1)        # Cargo en v0 buffer -> sizeBytes.
                sizeBytes es el tercer elemento
807                                 # del struct Buffer.
808         b       verifyMemory    # Salto incondicional para verificar
                 la asignacion de memoria.
809 compareQuantities:
810         # (buffer->quantityCharactersInBuffer >= buffer->sizeBytes)
811         lw      v0,60($fp)      # Cargo en v0 la direccion de buffer.
812         lw      v1,60($fp)      # Cargo en v1 la direccion de buffer.
813         lw      a0,4(v0)        # Cargo en a0 buffer->
                quantityCharactersInBuffer. En el struct
814                                 # Buffer quantityCharactersInBuffer
                                        es el segundo elemento.
815         lw      v0,8(v1)        # Cargo en v0 buffer->sizeBytes. En
                el struct
816                                 # Buffer sizeBytes es el tercer
                                        elemento.
817         sltu    v0,a0,v0        # Guardo en v0 TRUE si buffer->
                quantityCharactersInBuffer es mas
818                                 # chico que buffer->sizeBytes, sino
                                        guardo FALSE (=0).
819         bne     v0,FALSE,verifyMemory # If (buffer->
                quantityCharactersInBuffer < buffer->sizeBytes)
820                                   # goto verifyMemory
821
822         # buffer->quantityCharactersInBuffer is >= than buffer->
                sizeBytes
823
824         # size_t bytesLexicoPreview = buffer->sizeBytes;
825         lw      v0,60($fp)      # Cargo en v0 la direccion de buffer.
826         lw      v0,8(v0)        # Cargo en v0 buffer->sizeBytes. En
                el struct
827                                 # Buffer sizeBytes es el tercer
                                        elemento.
828         sw      v0,28($fp)      # Cargo en la direccion de memoria
                28($fp) el valor de
829                                 # buffer->sizeBytes, que
                                        representaria la variabla
                                        bytesLexicoPreview.
830
831         # buffer->sizeBytes = bytesLexicoPreview * 2;
832         lw      v1,60($fp)      # Cargo en v1 la direccion de buffer.
833         lw      v0,28($fp)      # Cargo en v0 bytesLexicoPreview.
834         sll     v0,v0,1         # bytesLexicoPreview * 2 y guardo
                resultado en v0.
```

```
835                                 # 1 porque 2 elevado a 1 es igual a
                                        2.
836             sw      v0,8(v1)        # Guardo el resultado de la
                    multiplicacion en buffer->sizeBytes.
837
838             # char * auxiliary = myRealloc(buffer->buffer, buffer->
                    sizeBytes*sizeof(char), bytesLexicoPreview);
839             lw      v0,60($fp)      # Cargo en v0 la direccion de buffer
840             lw      v1,60($fp)      # Cargo en v1 la direccion de buffer
841             lw      a0,0(v0)        # Cargo en a0 buffer->buffer.
                    Parametro para myRealloc.
842             lw      a1,8(v1)        # Cargo en a1 buffer->sizeBytes.
                    Parametro para myRealloc.
843             lw      a2,28($fp)      # Cargo en a2 bytesLexicoPreview.
                    Parametro para myRealloc.
844             la      t9,myRealloc
845             jal     ra,t9           # Ejecuto la funcion myRealloc.
846             sw      v0,32($fp)      # Guardo en la direccion 32($fp) la
                    memoria reservada.
847
848             # (auxiliary == NULL)
849             lw      v0,32($fp)      # Cargo en v0 la memoria reservada.
850             bne     v0,zero,memoryAllocation # If (auxiliary != NULL)
                    goto memoryAllocation.
851
852             # Hubo problemas con la reasignacion de memoria.
853             lw      a0,60($fp)      # Cargo en a0 buffer. Parametro de la
                     funcion cleanContentBuffer.
854             la      t9,cleanContentBuffer
855             jal     ra,t9           # Ejecuto la funcion
                    cleanContentBuffer para liberar memoria.
856
857             b       verifyMemory
858     memoryAllocation:
859             # buffer->buffer = auxiliary;
860             lw      v1,60($fp)
861             lw      v0,32($fp)
862             sw      v0,0(v1)
863     verifyMemory:
864             # (buffer->buffer == NULL)
865             lw      v0,60($fp)
866             lw      v0,0(v0)
867             bne     v0,zero,loadCharacterInBuffer # If (buffer->buffer
                    != NULL) goto loadCharacterInBuffer
868
869             # buffer->buffer is NULL => Mensaje de error
870             li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
                    FILE_DESCRIPTOR_STDERR.
871             la      a1,MENSAJE_ERROR_MEMORIA_BUFFER # Cargo en a1 la
                    direccion de memoria donde se encuentra el mensaje a
                    cargar.
872             li      a2,BYTES_MENSAJE_ERROR_MEMORIA_BUFFER # Cargo en a2
                    la cantidad de bytes a escribir.
873             li      v0, SYS_write
874             syscall   # No controlo error porque sale de por si de la
                    funcion por error.
875
876             # return ERROR_MEMORY;
877             li      v0,ERROR_MEMORY
878             sw      v0,36($fp)      # Guardo en la direccion 36($fp) el
                    codigo de error,
879                                     # resultado de la funcion.
880             b       returnLoadInBuffer
881     loadCharacterInBuffer:
882             # buffer->buffer[buffer->quantityCharactersInBuffer] =
                    character;
883             lw      v0,60($fp)
884             lw      v1,60($fp)
885             lw      a0,0(v0)        # Cargo en a0 buffer->buffer
886             lw      v0,4(v1)        # Cargo en v0 buffer->
                    quantityCharactersInBuffer
887             addu    v1,a0,v0        # Corrimiento sobre buffer->buffer.
                    Guardo en v1
888             lbu     v0,24($fp)      # Carga character en v0
```

```
889         sb      v0,0(v1)        # Asigno character a esa direccion de
                memoria.
890
891         # buffer->quantityCharactersInBuffer ++;
892         lw      v1,60($fp)
893         lw      v0,60($fp)
894         lw      v0,4(v0)
895         addu    v0,v0,1
896         sw      v0,4(v1)
897
898         # return OKEY;
899         sw      zero,36($fp)    # OKEY = 0
900 returnLoadInBuffer:
901         lw      v0,36($fp)
902         move    sp,$fp
903         lw      ra,52(sp)
904         lw      $fp,48(sp)
905         lw      s0,40(sp)
906         # destruyo stack frame
907         addu    sp,sp,56
908         # vuelvo a funcion llamante
909         j       ra
910
911         .end    loadInBuffer
912
913
914 ##------ cleanContentBuffer ------##
915
916         .align  2
917         .globl  cleanContentBuffer
918         .ent    cleanContentBuffer
919 cleanContentBuffer:
920         .frame  $fp,40,ra
921         .set    noreorder
922         .cpload t9
923         .set    reorder
924
925         # Stack frame creation
926         subu    sp,sp,40
927
928         .cprestore 16
929         sw      ra,32(sp)
930         sw      $fp,28(sp)
931         sw      gp,24(sp)
932
933         # de aqui al fin de la funcion uso $fp en lugar de sp.
934         move    $fp,sp
935
936         # Parametro
937         sw      a0,40($fp)      # Guardo en la direccion 40($fp) el
                parametro * buffer (Buffer * buffer).
938
939         # (buffer->buffer != NULL)
940         lw      v0,40($fp)
941         lw      v0,0(v0)        # Cargo en v0 buffer -> buffer
942         beq     v0,zero,cleanQuantities # If (buffer->buffer == NULL
                ) goto cleanQuantities
943
944         # buffer->buffer is not NULL
945
946         # free(buffer->buffer);
947         lw      v0,40($fp)
948         lw      a0,0(v0)
949         la      t9,myfree
950         jal     ra,t9
951
952         # buffer->buffer = NULL;
953         lw      v0,40($fp)
954         sw      zero,0(v0)
955 cleanQuantities:
956         # buffer->quantityCharactersInBuffer = 0;
957         lw      v0,40($fp)
958         sw      zero,4(v0)
959
```

```
960         # buffer->sizeBytes = 0;
961         lw      v0,40($fp)
962         sw      zero,8(v0)
963
964         move    sp,$fp
965         lw      ra,32(sp)
966         lw      $fp,28(sp)
967         # destruyo stack frame
968         addu    sp,sp,40
969         # vuelvo a funcion llamante
970         j       ra
971
972         .end    cleanContentBuffer
973
974
975 #   ------------------------------------------------------------------------
        #
976
977 ## Variables auxiliares
978
979         .data
980
981
982         # ---------------------------------------- #
983         #                                          #
984         # typedef struct  {                        #
985         #    char * buffer;                        #
986         #    int quantityCharactersInBuffer;       #
987         #    size_t sizeBytes;                     #
988         # } Buffer;                                #
989         #                                          #
990         # Buffer ibuffer                           #
991         # Buffer obuffer                           #
992         #                                          #
993         # ---------------------------------------- #
994
995
996         ## Variables para la parte de input
997
998         .globl  ifd
999 #       .section        .bss    #TODO DESPUES VER SI ESTO SE PUEDE
        ELIMINAR
1000        .align  2
1001        .type   ifd, object.size ifd, 4ifd:.space 4.globl lastPositionInIBufferRead.align
                2.type lastPositionInIBufferRead, object
1002        .size   lastPositionInIBufferRead, 4
1003 lastPositionInIBufferRead:
1004        .word   -1
1005
1006        .globl  ibuffer
1007 #       .section        .bss  TODO VER SI ANDA BIEN Y SACARLO
1008        .align  2
1009        .type   ibuffer, object.size ibuffer, 12ibuffer:.space 12.globl endIFile.globl
                endIFile.align 2.type endIFile, object
1010        .size   endIFile, 4
1011 endIFile:
1012        .space  4
1013
1014
1015        ## Variables para la parte de input
1016
1017        .globl  ofd
1018        .align  2
1019        .type   ofd, object.size ofd, 4ofd:.space 4.globl obuffer.align 2.type obuffer,
                object
1020        .size   obuffer, 12
1021 obuffer:
1022        .space  12
1023
1024 ## Mensajes de error
1025
1026        .rdata
1027
```

```
1028            .align  2
1029    MENSAJE_ERROR_MEMORIA_IBUFFER:
1030            .ascii  "[Error] Hubo un error de asignacion de memoria (
                    ibuffer)"
1031            .ascii  ". \n\000"
1032
1033            .align  2
1034    MENSAJE_ERROR_LECTURA_ARCHIVO:
1035            .ascii  "[Error] Hubo un error en la lectura de datos del
                    archivo"
1036            .ascii  ". \n\000"
1037
1038            .align  2
1039    MENSAJE_ERROR_ESCRITURA_ARCHIVO:
1040            .ascii  "[Error] Hubo un error al escribir en el archivo. \n
                    \000"
1041
1042             .align 2
1043    MENSAJE_ERROR_MEMORIA_OBUFFER:
1044            .ascii  "[Error] Hubo un error de asignacion de memoria (
                    obuffer)"
1045            .ascii  ". \n\000"
1046
1047            .align  2
1048    MENSAJE_ERROR_MEMORIA_BUFFER:
1049            .ascii  "[Error] Hubo un error de asignacion de memoria (
                    buffer)"
1050            .ascii  ". \n\000"
```

Stack frame:

void initializeOutput(int oFileDescriptor, size_t obytes)		
Offset	**Contents**	**Type reserved area**
12	obytes	**ABA (caller)**
8	oFileDescriptor	
4	fp	**SRA**
0	gp	
Stack frame: initializeOutput		

Figura 2: Stack frame: initializeOutputSF

Stack frame:

void initializeInput(int iFileDescriptor, size_t ibytes)		
Offset	**Contents**	**Type reserved area**
12	ibytes	**ABA (caller)**
8	iFileDescriptor	
4	fp	**SRA**
0	gp	
Stack frame: initializeInput		

Figura 3: Stack frame: initializeInputSF

Stack frame:

int getch()		
Offset	**Contents**	**Type reserved area**
36	////////////////////////////////	**SRA**
32	ra	
28	fp	
24	gp	
20	Resultado de la funció	**LTA**
16	resultLoadIBuffer	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: getch		

Figura 4: Stack frame: getchSF

Stack frame:

void freeResources()		
Offset	**Contents**	**Type reserved area**
28	////////////////////////////////	**SRA**
24	ra	
20	fp	
16	gp	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: freeResources		

Figura 5: Stack frame: freeResourcesSF

Stack frame:

int flush()		
Offset	**Contents**	**Type reserved area**
36	////////////////////////////////	**SRA**
32	ra	
28	fp	
24	gp	
20	////////////////////////////////	**LTA**
16	Resultado de la función	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: flush		

Figura 6: Stack frame: flushSF

Stack frame:

void cleanContentBuffer(Buffer * buffer)		
Offset	**Contents**	**Type reserved area**
32	* buffer	**ABA (caller)**
28	////////////////////////////////	**SRA**
24	ra	
20	fp	
16	gp	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: cleanContentBuffer		

Figura 7: Stack frame: cleanContentBufferSF

Stack frame:

int loadIBufferWithIFile()		
Offset	**Contents**	**Type reserved area**
44	////////////////////////////////	**SRA**
40	ra	
36	fp	
32	gp	
28	Resultado de la funció	**LTA**
24	bytesRead	
20	bytesToRead	
16	completeDelivery	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: loadIBufferWithIFile		

Figura 8: Stack frame: loadIBufferWithIFileSF

Stack frame:

int loadInBuffer(char character, Buffer * buffer, size_t sizeInitial)		
Offset	**Contents**	**Type reserved area**
52	sizeInitial	**ABA (caller)**
48	* buffer	
44	ra	**SRA**
40	fp	
36	gp	
32	s0	
28	Resultado de la función	**LTA**
24	auxiliary	
20	bytesLexicoPreview	
16	character	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: loadInBuffer		

Figura 9: Stack frame: loadInBufferSF

Stack frame:

int putch(int character)		
Offset	**Contents**	**Type reserved area**
40	character	**ABA (caller)**
36	////////////////////////////////	**SRA**
32	ra	
28	fp	
24	gp	
20	////////////////////////////////	**LTA**
16	Resultado de la funció	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: putch		

Figura 10: Stack frame: putchSF

Stack frame:

int writeBufferInOFile()		
Offset	**Contents**	**Type reserved area**
52	////////////////////////////////	**SRA**
48	ra	
44	fp	
40	gp	
36	////////////////////////////////	**LTA**
32	Resultado de la funció	
28	bytesWrite	
24	bytesToWrite	
20	bytesWriteAcum	
16	completeDelivery	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: writeBufferInOFile		

Figura 11: Stack frame: writeBufferInOFileSF

4.2. Código MIPS32: palindromeFunctions.S

```
#include <mips/regdef.h>
#include <sys/syscall.h>

#include "constants.h"
##include "bufferFunctions.h"
##include "memoryFunctions.h"

# Size mensajes
#define BYTES_MENSAJE_ERROR_PUTCH    65


##------ toLowerCase ------##

        .text
        .align  2
        .globl  toLowerCase
        .ent    toLowerCase
toLowerCase:
        .frame  $fp,24,ra
        .set    noreorder
        .cpload t9
        .set    reorder

        # Stack frame creation
        subu    sp,sp,24

        .cprestore 0
        sw      $fp,20(sp)
        sw      gp,16(sp)

        # de aqui al fin de la funcion uso $fp en lugar de sp.
        move    $fp,sp

        # Parametro
        sb      a0,8($fp)       # Guardo en la direccion 8($fp) el
            contenido de
                                # a0 que es word (char word).

        # (word >= 65 && word <= 90)

        # (word >= 65)
        lb      v0,8($fp)
        slt     v0,v0,65      # Guardo en v0 TRUE si word es mas
            chico que 65, sino guardo FALSE.
        bne     v0,FALSE,returnToLowerCase  # If (word < 65) goto
            returnToLowerCase

        # word is >= 65

        # (word <= 90)
        lb      v0,8($fp)
        slt     v0,v0,91      # Guardo en v0 TRUE si word es mas
            chico que 91, sino guardo FALSE.
        beq     v0,FALSE,returnToLowerCase # If (word >= 91) goto
            returnToLowerCase

        # word is <= 90

        # word += 32;
        lbu     v0,8($fp)
        addu    v0,v0,32
        sb      v0,8($fp)  # Guardo en la direccion 8($fp) el
            resultado de la funcion,
                           # que coincide con la variable word.
returnToLowerCase:
        lb      v0,8($fp)
        move    sp,$fp
        lw      $fp,20(sp)
        # destruyo stack frame
        addu    sp,sp,24
```

```
65          # vuelvo a funcion llamante
66          j       ra
67
68          .end    toLowerCase
69
70
71
72  ##------ verifyPalindromic ------##
73
74          .align  2
75          .globl  verifyPalindromic
76          .ent    verifyPalindromic
77  verifyPalindromic:
78          .frame  $fp,72,ra
79          .set    noreorder
80          .cpload t9
81          .set    reorder
82
83          subu    sp,sp,72
84
85          .cprestore 16
86          sw      ra,64(sp)
87          sw      $fp,60(sp)
88          sw      gp,56(sp)
89
90          move    $fp,sp
91
92          # (lexico.buffer == NULL || lexico.
                quantityCharactersInBuffer <= 0)
93
94          # (lexico.buffer == NULL)
95          lw      v0,lexico
96          beq     v0,zero,returnFalse # If (lexico.buffer == NULL)
                goto returnFalse
97
98          # lexico.buffer is not NULL
99
100         # (lexico.quantityCharactersInBuffer <= 0)
101         lw      v0,lexico+4
102         blez    v0,returnFalse   # If (lexico.
                quantityCharactersInBuffer <= 0) returnFalse
103
104         # lexico.quantityCharactersInBuffer is > 0
105         b       verifyOneCharacter
106 returnFalse:
107         # return FALSE;
108         sw      zero,48($fp)     # FALSE = 0, guardo en la direccion
                 48($fp) el resultado de la funcion
109         b       returnVerifyPalindromic
110 verifyOneCharacter:
111         # (lexico.quantityCharactersInBuffer == 1)
112         lw      v1,lexico+4
113         li      v0,1
114         bne     v1,v0,compareCharacters # IF (lexico.
                quantityCharactersInBuffer != 1)
115                                         # goto compareCharacters
116
117         # lexico.quantityCharactersInBuffer is equals to 1
118
119         # return TRUE;
120         li      v0,TRUE
121         sw      v0,48($fp)    # guardo en la direccion 48($fp) el
                resultado de la funcion
122         b       returnVerifyPalindromic
123 compareCharacters:
124         # double middle = (double)lexico.quantityCharactersInBuffer
                / 2;
125         l.s     $f0,lexico+4  # Cargo lexico.
                quantityCharactersInBuffer en f0
126         cvt.d.w $f2,$f0       # Convierto el integer
                quantityCharactersInBuffer a double
127         l.d     $f0,doubleWord   # Cargo en f0 el valor 2.
128         div.d   $f0,$f2,$f0     # Division con Double (double)
                quantityCharacterInWord / 2;
```

```
129                                 # Sintaxis: div.d FRdest, FRsrc1,
                                        FRsrc2
130             s.d     $f0,24($fp)     # Guarda el resultado de la division
                    en 24($fp). O sea,
131                                 # middle (double middle = (double)
                                        quantityCharacterInWord / 2;)
132
133             # int idx = 0;
134             sw      zero,32($fp)    # En 32($fp) se encuentra idx (int
                    idx = 0;).
135
136             # int validPalindromic = TRUE;
137             li      v0,TRUE
138             sw      v0,36($fp)    # En 36($fp) esta la variable
                    validPalindromic.
139
140             # int last = lexico.quantityCharactersInBuffer - 1;
141             lw      v0,lexico+4
142             addu    v0,v0,-1
143             sw      v0,40($fp)    # En 40($fp) esta la variable last.
144     whileMirror:
145             # (idx < middle && last >= middle && validPalindromic ==
                    TRUE)
146             l.s     $f0,32($fp)   # Cargo idx en f0.
147             cvt.d.w $f2,$f0       # Convierto el integer idx a double y
                    lo guardo en
148                                 # f2 para poder hacer la comparacion.
149             l.d     $f0,24($fp)   # Cargo en a0 la variable middle.
150             c.lt.d  $f2,$f0       # Compara la variable idx con la
                    variable middle, y
151                                 # setea el condition flag en true si
                                        el primero (idx) es
152                                 # mas chico que el segundo (middle).
153             bc1t    verifyConditionLastWithMiddle # Si el condition flag
                     es true, continua
154                                                   # haciendo las
                                                          comparaciones.
155             b       whileMirrorFinalized   # Si el condition flag es
                    false, salta al final de la
156                                       # funcion, devolviendo el
                                              valor de la variable
                                              validPalindromic.
157     verifyConditionLastWithMiddle:
158             l.s     $f0,40($fp)  # Cargo la variable last en f0.
159             cvt.d.w $f2,$f0      # Convierto el integer last a double y
                    lo guardo
160                                 # en f2 para poder hacer la comparacion
                                        .
161             l.d     $f0,24($fp)  # Cargo en f0 el contenido de la
                    variable middle.
162             c.le.d  $f0,$f2      # Compara el contenido de la variable
                    last con la variable
163                                 # middle, y setea el condition flag en
                                        true si
164                                 # middle es menor o igual a last, sino
                                        false
165             bc1t    verifyConditionPalindromicTrue # Si el condition
                    flag es true,
166                                                    # continua haciendo
                                                           las comparaciones
                                                           .
167             b       whileMirrorFinalized  # false
168     verifyConditionPalindromicTrue:
169             lw      v1,36($fp)  # Carga en v1 validPalindromic
170             li      v0,TRUE
171             beq     v1,v0,whileMirrorContent  # If (validPalindromic ==
                    TRUE) goto whileMirrorContent
172             b       whileMirrorFinalized
173     whileMirrorContent:
174             # char firstCharacter = toLowerCase(lexico.buffer[idx]);
175             lw      v1,lexico
176             lw      v0,32($fp)
177             addu    v0,v1,v0
178             lb      v0,0(v0)
```

```
179         move    a0,v0
180         la      t9,toLowerCase
181         jal     ra,t9
182         sb      v0,44($fp)   # Guardo en la direccion 44($fp) la
                variable firstCharacter
183
184         # char lastCharacter = toLowerCase(lexico.buffer[last]);
185         lw      v1,lexico
186         lw      v0,40($fp)
187         addu    v0,v1,v0
188         lb      v0,0(v0)
189         move    a0,v0
190         la      t9,toLowerCase
191         jal     ra,t9
192         sb      v0,45($fp)   # Guardo en la direccion 45($fp) la
                variable lastCharacter
193
194         # (firstCharacter != lastCharacter)
195         lb      v1,44($fp)   # Cargo firstCharacter en v1
196         lb      v0,45($fp)   # Cargo lastCharacter en v0
197         beq     v1,v0,continueWhile # If (firstCharacter ==
                lastCharacter) goto continueWhile
198
199         # firstCharacter != lastCharacter
200
201         # validPalindromic = FALSE;
202         sw      zero,36($fp)
203 continueWhile:
204         # idx ++;
205         lw      v0,32($fp)
206         addu    v0,v0,1
207         sw      v0,32($fp)
208
209         # last --;
210         lw      v0,40($fp)
211         addu    v0,v0,-1
212         sw      v0,40($fp)
213
214         b       whileMirror
215 whileMirrorFinalized:
216         # return validPalindromic;
217         lw      v0,36($fp)   # Cargo el contenido de
                validPalindromic en v0
218         sw      v0,48($fp)   # Guardo en la direccion 48($fp) el
                resultado de la funcion
219                              # que esta en v0.
220 returnVerifyPalindromic:
221         lw      v0,48($fp)
222         move    sp,$fp
223         lw      ra,64(sp)
224         lw      $fp,60(sp)
225         # destruyo stack frame
226         addu    sp,sp,72
227         # vuelvo a funcion llamante
228         j       ra
229
230         .end    verifyPalindromic
231
232
233
234 ##------ isKeywords ------##
235
236         .align  2
237         .globl  isKeywords
238         .ent    isKeywords
239 isKeywords:
240         .frame  $fp,24,ra
241         .set    noreorder
242         .cpload t9
243         .set    reorder
244
245         # Stack frame creation
246         subu    sp,sp,24
247
```

```
248         .cprestore 0
249         sw      $fp,20(sp)
250         sw      gp,16(sp)
251
252         # de aqui al fin de la funcion uso $fp en lugar de sp.
253         move    $fp,sp
254
255         # Parametro
256         sb      a0,8($fp)       # Guardo en la direccion 8($fp) el
                parametro character
257                                 # que viene en a0 (char character).
258
259         # ((character >= 65 && character <= 90) || (character >= 97
                && character <= 122)
260         #               || (character >= 48 && character <= 57)
261         #               || character == 45 || character == 95)
262
263
264         # (character >= 65 && character <= 90)
265
266         # (character >= 65)   ---   A - Z = [65 - 90]
267         lb      v0,8($fp)
268         slt     v0,v0,65        # Guarda en v0 TRUE si character es
                mas chico que 65, sino FALSE.
269         bne     v0,FALSE,verifyCharacterOfaToz # Si no es igual a
                FALSE, o sea, character < 65,
270                                                 # salta a
                                                        VerifyCharacterOfaToz
                                                        .
271
272         # (character >= 65 && character <= 90)
273         lb      v0,8($fp)
274         slt     v0,v0,91      # Compara el contenido de la variable
                character con el
275                               # literal 91, y guarda true en v0 si el
                                      primero (character)
276                               # es mas chico que el segundo (91).
277         bne     v0,FALSE,returnIsKeywordsTrue # Si no es igual a
                FALSE, o sea,
278                                                 # character < 91,
                                                        salta a
                                                        ReturnIsKeywordsTrue
                                                        .
279 verifyCharacterOfaToz:
280         # (character >= 97 && character <= 122)      ---   a - z =
                [97 - 122]
281
282         # (character >= 97)
283         lb      v0,8($fp)
284         slt     v0,v0,97    # Guarda true en v0 si el primero (
                character) es mas
285                             # chico que el segundo (97).
286         bne     v0,FALSE,verifyCharacterOf0To9 # Si no es igual a
                FALSE, o sea,
287                                                 # character < 97,
                                                        salta a
                                                        verifyCharacterOf0To9
                                                        .
288
289         # (character <= 122)
290         lb      v0,8($fp)
291         slt     v0,v0,123   # Guarda true en v0 si el primero (
                character)
292                             # es mas chico que el segundo (123).
293         bne     v0,FALSE,returnIsKeywordsTrue # Si no es igual a
                FALSE, o sea,
294                                                 # character < 123,
                                                        salta a
                                                        returnIsKeywordsTrue
                                                        .
295 verifyCharacterOf0To9:
296         # (character >= 48 && character <= 57)      ---   0 - 9 =
                [48 - 57]
297
```

```
298         # (character >= 48)
299         lb      v0,8($fp)
300         slt     v0,v0,48
301         bne     v0,FALSE,verifyCharacterGuionMedio
302
303         # (character <= 57)
304         lb      v0,8($fp)
305         slt     v0,v0,58
306         bne     v0,FALSE,returnIsKeywordsTrue
307 verifyCharacterGuionMedio:
308         # character == 45
309         lb      v1,8($fp)
310         li      v0,45
311         beq     v1,v0,returnIsKeywordsTrue
312
313         # character == 95
314         lb      v1,8($fp)
315         li      v0,95
316         beq     v1,v0,returnIsKeywordsTrue
317
318         b       returnIsKeywordsFalse
319 returnIsKeywordsTrue:
320         li      v0,TRUE
321         sw      v0,12($fp)      # Guardo en la direccion 12($fp) el
                resultado
322                                 # de la funcion, en este caso TRUE.
323         b       returnIsKeywords
324 returnIsKeywordsFalse:
325         sw      zero,12($fp)    # Guardo en la direccion 12($fp) el
                resultado
326                                 # de la funcion, en este caso FALSE.
327 returnIsKeywords:
328         lw      v0,12($fp)
329         move    sp,$fp
330         lw      $fp,20(sp)
331         # destruyo stack frame
332         addu    sp,sp,24
333         # vuelvo a funcion llamante
334         j       ra
335
336         .end    isKeywords
337
338
339
340 ##------ saveIfPalindrome ------##
341
342         .align  2
343         .globl  saveIfPalindrome
344         .ent    saveIfPalindrome
345 saveIfPalindrome:
346         .frame  $fp,64,ra
347         .set    noreorder
348         .cpload t9
349         .set    reorder
350
351         # Stack frame creation
352         subu    sp,sp,64
353
354         .cprestore 16
355         sw      ra,56(sp)
356         sw      $fp,52(sp)
357         sw      gp,48(sp)
358
359         # de aqui al fin de la funcion uso $fp en lugar de sp.
360         move    $fp,sp
361
362         # int itsPalindromic = verifyPalindromic();
363         la      t9,verifyPalindromic
364         jal     ra,t9
365         sw      v0,24($fp)    # Guardo en la direccion 24($fp)
                itsPalindromic.
366
367         # (itsPalindromic == TRUE)
368         lw      v1,24($fp)
```

```
369         li      v0,TRUE
370         bne     v1,v0,returnOkeySaveIfPalindrome # If (
                itsPalindromic != TRUE) goto
371                                                  #
                                                         returnOkeySaveIfPalindrome
372
373         # int idx = 0;
374         sw      zero,28($fp)
375
376         # int error = FALSE;
377         sw      zero,32($fp)
378 whilePutch:
379         # (idx < lexico.quantityCharactersInBuffer && error == FALSE
                )
380
381         # (idx < lexico.quantityCharactersInBuffer)
382         lw      v0,28($fp)      # Cargo en v0 idx
383         lw      v1,lexico+4     # Cargo en v1
                quantityCharactersInBuffer
384         slt     v0,v0,v1        # Cargo en v0 TRUE si idx <
                quantityCharactersInBuffer, sino FALSE
385         beq     v0,FALSE,loadLineJump # If (idx >=
                quantityCharactersInBuffer) goto loadLineJump
386
387         # (error == FALSE)
388         lw      v0,32($fp)
389         bne     v0,FALSE,loadLineJump # If (error != FALSE) goto
                loadLineJump
390
391         # int result = putch(lexico.buffer[idx]);
392         lw      v1,lexico
393         lw      v0,28($fp)
394         addu    v0,v1,v0
395         lb      v0,0(v0)
396         move    a0,v0
397         la      t9,putch
398         jal     ra,t9           # Ejecuto la funcion putch
399         sw      v0,36($fp)    # Guardo en la direccion 36($fp) el
                resultado de la funcion putch (result).
400
401         # (result == EOF_F)
402         lw      v1,36($fp)
403         li      v0,EOF_F
404         bne     v1,v0,incrementIdx  # If (result != EOF) goto
                incrementIdx
405
406 #       # error = TRUE;
407         li      v0,TRUE
408         sw      v0,32($fp)      # Guardo TRUE en la variable error.
409 incrementIdx:
410         # idx ++;
411         lw      v0,28($fp)
412         addu    v0,v0,1
413         sw      v0,28($fp)
414
415         b       whilePutch
416 loadLineJump:
417         # (error == FALSE)
418         lw      v0,32($fp)
419         bne     v0,FALSE,returnWithError  # If (error != FALSE) goto
                 returnWithError
420
421         # int result = putch('\n');
422         li      a0,10           # '\n' = 10
423         la      t9,putch
424         jal     ra,t9
425         sw      v0,36($fp)
426
427         # (result == EOF_F)
428         lw      v1,36($fp)
429         li      v0,EOF_F
430         bne     v1,v0,returnWithError
431
```

```
#       # error = TRUE;
        li      v0,TRUE
        sw      v0,32($fp)
returnWithError:
        # (error == TRUE)
        lw      v1,32($fp)
        li      v0,TRUE
        bne     v1,v0,returnOkeySaveIfPalindrome # If (error != TRUE
            ) goto returnOkeySaveIfPalindrome

        # Mensaje de error
        li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
            FILE_DESCRIPTOR_STDERR.
        la      a1,MENSAJE_ERROR_PUTCH  # Cargo en a1 la direccion
            de memoria donde se encuentra el mensaje a cargar.
        li      a2,BYTES_MENSAJE_ERROR_PUTCH    # Cargo en a2 la
            cantidad de bytes a escribir.
        li      v0, SYS_write
        syscall    # No controlo error porque sale de por si de la
            funcion por error.

        # return ERROR_PUTCH;
        li      v0,ERROR_PUTCH
        sw      v0,40($fp)
        b       returnSaveIfPalindrome
returnOkeySaveIfPalindrome:
        sw      zero,40($fp)    # OKEY = 0
returnSaveIfPalindrome:
        lw      v0,40($fp)
        move    sp,$fp
        lw      ra,56(sp)
        lw      $fp,52(sp)
        # destruyo stack frame
        addu    sp,sp,64
        # vuelvo a funcion llamante
        j       ra

        .end    saveIfPalindrome



#------ palindrome ------##

        .align  2
        .globl  palindrome
        .ent    palindrome
palindrome:
        .frame  $fp,56,ra
        .set    noreorder
        .cpload t9
        .set    reorder

        # Stack frame creation
        subu    sp,sp,56

        .cprestore 16
        sw      ra,48(sp)
        sw      $fp,44(sp)
        sw      gp,40(sp)

        # de aqui al fin de la funcion uso $fp en lugar de sp.
        move    $fp,sp

        # Parametros
        sw      a0,56($fp)      # Guardo en la direccion 56($fp) ifd
             que estaba en a0.
        sw      a1,60($fp)      # Guardo en la direccion 60($fp)
            ibytes que estaba en a1.
        sw      a2,64($fp)      # Guardo en la direccion 64($fp) ofd
             que estaba en a2.
        sw      a3,68($fp)      # Guardo en la direccion 68($fp)
            obytes que estaba en a3.

        # initializeInput(ifd, ibytes);
```

```
497         lw      a0,56($fp)
498         lw      a1,60($fp)
499         la      t9,initializeInput
500         jal     ra,t9
501
502         # initializeOutput(ofd, obytes);
503         lw      a0,64($fp)
504         lw      a1,68($fp)
505         la      t9,initializeOutput
506         jal     ra,t9
507
508         # lexico.quantityCharactersInBuffer = 0;
509         sw      zero,lexico+4
510
511         # int icharacter = getch();
512         la      t9,getch
513         jal     ra,t9
514         sw      v0,24($fp)      # Guardo en la direccion 4($fp)
                icharacter,
515                                 # resultado de la funcion getch().
516
517         # int result = OKEY;
518         sw      zero,28($fp)    # Guardo en la direccion 28($fp)
                OKEY (= 0).
519 whilePalindrome:
520         # (icharacter != EOF && icharacter != ERROR_I_READ && result
                == OKEY)
521
522         # (icharacter != EOF_F)
523         lw      v1,24($fp)
524         li      v0,EOF_F
525         beq     v1,v0,flushLexico # If (icharacter == EOF) goto
                flushLexico
526
527         # (icharacter != ERROR_I_READ)
528         lw      v1,24($fp)
529         li      v0,ERROR_I_READ
530         beq     v1,v0,flushLexico # If (icharacter == ERROR_I_READ)
                goto flushLexico
531
532         # (result == OKEY)
533         lw      v0,28($fp)
534         bne     v0,OKEY,flushLexico # If (result != OKEY) goto
                flushLexico
535
536         # In while
537
538         # char character = icharacter;
539         lbu     v0,24($fp)
540         sb      v0,32($fp)    # Guardo en la direccion 32($fp)
                character
541
542         # (isKeywords(character) == TRUE)
543         lb      v0,32($fp)
544         move    a0,v0
545         la      t9,isKeywords
546         jal     ra,t9
547         move    v1,v0
548         li      v0,TRUE
549         bne     v1,v0,isNotKeyword # If (resultado de isKeywords !=
                TRUE) goto isNotKeyword
550
551         # is keyword
552
553         # result = loadInBuffer(character, &lexico,
                LEXICO_BUFFER_SIZE);
554         lb      v0,32($fp)
555         move    a0,v0
556         la      a1,lexico
557         li      a2,LEXICO_BUFFER_SIZE
558         la      t9,loadInBuffer
559         jal     ra,t9         # Ejecuto la funcion loadInBuffer.
560         sw      v0,28($fp)    # Cargo en la direccion 28($fp) el
                resultado de la funcion loadInBuffer.
```

```
561
562             b       loadNextCharacter
563     isNotKeyword:
564             # result = saveIfPalindrome();
565             la      t9,saveIfPalindrome
566             jal     ra,t9
567             sw      v0,28($fp)
568
569             # cleanContentBuffer(&lexico);
570             la      a0,lexico
571             la      t9,cleanContentBuffer
572             jal     ra,t9
573     loadNextCharacter:
574             # icharacter = getch();
575             la      t9,getch
576             jal     ra,t9
577             sw      v0,24($fp)      # icharacter
578
579             b       whilePalindrome
580     flushLexico:
581             # int resultFlush = saveIfPalindrome();
582             la      t9,saveIfPalindrome
583             jal     ra,t9
584             sw      v0,36($fp)      # Guardo en la direccion 36($fp) la
                    variable resultFlush,
585                                     # que representa el resultado de
                                            ejecutar la funcion
                                            saveIfPalindrome.
586
587             # (result == OKEY)
588             lw      v0,28($fp)
589             bne     v0,OKEY,cleanLexico # If (result != OKEY) goto
                    cleanLexico
590
591             # result = resultFlush;
592             lw      v0,36($fp)
593             sw      v0,28($fp)
594     cleanLexico:
595             # cleanContentBuffer(&lexico);
596             la      a0,lexico
597             la      t9,cleanContentBuffer
598             jal     ra,t9
599
600             # resultFlush = flush();
601             la      t9,flush
602             jal     ra,t9
603             sw      v0,36($fp)    # Guardo el resultado de flush en
                    resultFlush, 36($fp)
604
605             # (result == OKEY)
606             lw      v0,28($fp)
607             bne     v0,OKEY,continueWithFreeResources # If (result !=
                    OKEY) goto continueWithFreeResources
608
609             # result = resultFlush;
610             lw      v0,36($fp)
611             sw      v0,28($fp)
612     continueWithFreeResources:
613             # freeResources();
614             la      t9,freeResources
615             jal     ra,t9
616
617             lw      v0,28($fp)    # Cargo en v0 result que esta en la
                    direccion 28($fp).
618
619             move    sp,$fp
620             lw      ra,48(sp)
621             lw      $fp,44(sp)
622             # destruyo stack frame
623             addu    sp,sp,56
624             # vuelvo a funcion llamante
625             j       ra
626
627             .end    palindrome
```

```
628
629
630 #  ------------------------------------------------------------------------
      #
631
632 ## Variables auxiliares
633
634         .data
635
636         .globl  lexico
637         .align 2
638 lexico:
639         .space 12
640
641
642         .rdata
643         .align  3
644 doubleWord:
645         .word   0
646         .word   1073741824
647
648
649 ## Mensajes de error
650
651         .rdata
652
653         .align  2
654 MENSAJE_ERROR_PUTCH:
655         .ascii  "[Error] Error al escribir en el archivo output el
              palind"
656         .ascii  "romo. \n\000"
```

Stack frame:

int saveIfPalindrome()		
Offset	**Contents**	**Type reserved area**
52	////////////////////////////////	**SRA**
48	ra	
44	fp	
40	gp	
36	////////////////////////////////	**LTA**
32	Resultado de la función	
28	result	
24	error	
20	idx	
16	itsPalindromic	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: saveIfPalindrome		

Figura 12: Stack frame: savedIfPalindromeSF

Stack frame:

int isKeywords(char character)		
Offset	**Contents**	**Type reserved area**
16	character	**ABA (caller)**
12	fp	**SRA**
8	gp	
4	////////////////////////////////	**LTA**
0	Resultado de la función	

Stack frame: isKeywords

Figura 13: Stack frame: isKeywordsSF

Stack frame:

int verifyPalindromic()		
Offset	**Contents**	**Type reserved area**
60	////////////////////////////////	**SRA**
56	ra	
52	fp	
48	gp	
44	////////////////////////////////	**LTA**
40	Resultado de la función	
36	lastCharacter	
32	firstCharacter	
28	last	
24	validPalindromic	
20	idx	
16	middle	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	

Stack frame: verifyPalindromic

Figura 14: Stack frame: varifyPalindromicSF

Stack frame:

char toLowerCase(char word)		
Offset	**Contents**	**Type reserved area**
8	word	**ABA (caller)**
4	fp	**SRA**
0	gp	

Stack frame: toLowerCase

Figura 15: Stack frame: toLowerCaseSF

Stack frame:

int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes)		
Offset	**Contents**	**Type reserved area**
60	obytes	**ABA (caller)**
56	ofd	
52	ibytes	
48	ifd	
44	////////////////////////////////	**SRA**
40	ra	
36	fp	
32	gp	
28	resultFlush	**LTA**
24	character	
20	result	
16	icharacter	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	

Stack frame: palindrome

Figura 16: Stack frame: palindromoSF

4.3. Código MIPS32: memoryFunctions.S

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #define MYMALLOC_SIGNATURE 0xdeadbeef
5  #define MYMALLOC_SIGNATURE 0xdeadbeef
6
7  #ifndef PROT_READ
8  #define PROT_READ 0x01
9  #endif
10
11 #ifndef PROT_WRITE
12 #define PROT_WRITE 0x02
13 #endif
14
15 #ifndef MAP_PRIVATE
16 #define MAP_PRIVATE 0x02
17 #endif
18
19 #ifndef MAP_ANON
20 #define MAP_ANON 0x1000
21 #endif
22
23
24 ###------ myfree ------##
25
26          .globl  myfree
27          .ent    myfree
28 myfree:
29          subu    sp, sp, 40
30          sw      ra, 32(sp)
31          sw      $fp, 28(sp)
```

```
32          sw      a0, 24(sp)  # Temporary: argument pointer.
33          sw      a0, 20(sp)  # Temporary: actual mmap(2) pointer.
34          move    $fp, sp
35
36          # Calculate the actual mmap(2) pointer.
37          #
38          lw      t0, 24(sp)
39          subu    t0, t0, 8
40          sw      t0, 20(sp)
41
42          # XXX Sanity check: the argument pointer must be checked
43          # in before we try to release the memory block.
44          #
45          # First, check the allocation signature.
46          #
47          lw      t0, 20(sp) # t0: actual mmap(2) pointer.
48          lw      t1, 0(t0)
49          bne     t1, MYMALLOC_SIGNATURE, myfree_die
50
51          # Second, check the memory block trailer.
52          #
53          lw      t0, 20(sp) # t0: actual mmap(2) pointer.
54          lw      t1, 4(t0)  # t1: actual mmap(2) block size.
55          addu    t2, t0, t1 # t2: trailer pointer.
56          lw      t3, -4(t2)
57          xor     t3, t3, t1
58          bne     t3, MYMALLOC_SIGNATURE, myfree_die
59
60          # All checks passed. Try to free this memory area.
61          #
62          li      v0, SYS_munmap
63          lw      a0, 20(sp) # a0: actual mmap(2) pointer.
64          lw      a1, 4(a0)  # a1: actual allocation size.
65          syscall
66
67          # Bail out if we cannot unmap this memory block.
68          #
69          bnez    v0, myfree_die
70
71          # Success.
72          #
73          j myfree_return
74
75  myfree_die:
76          # Generate a segmentation fault by writing to the first
77          # byte of the address space (a.k.a. the NULL pointer).
78          #
79          sw t0, 0(zero)
80
81  myfree_return:
82          # Destroy the stack frame.
83          #
84          move    sp, $fp
85          lw      ra, 32(sp)
86          lw      $fp, 28(sp)
87          addu    sp, sp, 40
88
89          j       ra
90          .end    myfree
91
92
93
94  ##------ mymalloc ------##
95
96          .text
97          .align  2
98          .globl  mymalloc
99          .ent    mymalloc
100 mymalloc:
101         subu    sp, sp, 56
102         sw      ra, 48(sp)
103         sw      $fp, 44(sp)
104         sw      a0, 40(sp)  # Temporary: original allocation size.
105         sw      a0, 36(sp)  # Temporary: actual allocation size.
```

```
106         li      t0, -1
107         sw      t0, 32(sp)  # Temporary: return value (defaults to
                -1).
108 #if 0
109         sw      a0, 28(sp)  # Argument building area (#8 ).
110         sw      a0, 24(sp)  # Argument building area (#7 ).
111         sw      a0, 20(sp)  # Argument building area (#6).
112         sw      a0, 16(sp)  # Argument building area (#5).
113         sw      a0, 12(sp)  # Argument building area (#4, a3).
114         sw      a0,  8(sp)  # Argument building area (#3, a2).
115         sw      a0,  4(sp)  # Argument building area (#2, a1).
116         sw      a0,  0(sp)  # Argument building area (#1, a0).
117 #endif
118         move    $fp, sp
119
120         # Adjust the original allocation size to a 4-byte boundary.
121         #
122         lw      t0, 40(sp)
123         addiu   t0, t0, 3
124         and     t0, t0, 0xfffffffc
125         sw      t0, 40(sp)
126
127         # Increment the allocation size by 12 units, in order to
128         # make room for the allocation signature, block size and
129         # trailer information.
130         #
131         lw      t0, 40(sp)
132         addiu   t0, t0, 12
133         sw      t0, 36(sp)
134
135         # mmap(0, sz, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANON,
                -1, 0)
136         #
137         li      v0, SYS_mmap
138         li      a0, 0
139         lw      a1, 36(sp)
140         li      a2, PROT_READ|PROT_WRITE
141         li      a3, MAP_PRIVATE|MAP_ANON
142
143         # According to mmap(2), the file descriptor
144         # must be specified as -1 when using MAP_ANON.
145         #
146         li      t0, -1
147         sw      t0, 16(sp)
148
149         # Use a trivial offset.
150         #
151         li      t0, 0
152         sw      t0, 20(sp)
153
154         # XXX TODO.
155         #
156         sw      zero, 24(sp)
157         sw      zero, 28(sp)
158
159         # Excecute the syscall, save the return value.
160         #
161         syscall
162         sw      v0, 32(sp)
163         beqz    v0, mymalloc_return
164
165         # Success. Check out the allocated pointer.
166         #
167         lw      t0, 32(sp)
168         li      t1, MYMALLOC_SIGNATURE
169         sw      t1, 0(t0)
170
171         # The actual allocation size goes right after the signature.
172         #
173         lw      t0, 32(sp)
174         lw      t1, 36(sp)
175         sw      t1,  4(t0)
176
177         # Trailer information.
```

```
178         #
179         lw      t0, 36(sp) # t0: actual allocation size.
180         lw      t1, 32(sp) # t1: Pointer.
181         addu    t1, t1, t0 # t1 now points to the trailing 4-byte
                area.
182         xor     t2, t0, MYMALLOC_SIGNATURE
183         sw      t2, -4(t1)
184
185         # Increment the result pointer.
186         #
187         lw      t0, 32(sp)
188         addiu   t0, t0, 8
189         sw      t0, 32(sp)
190
191 mymalloc_return:
192         # Restore the return value.
193         #
194         lw      v0, 32(sp)
195
196         # Destroy the stack frame.
197         #
198         move    sp, $fp
199         lw              ra, 48(sp)
200         lw              $fp, 44(sp)
201         addu    sp, sp, 56
202
203         j               ra
204         .end    mymalloc
205
206
207
208 #######  myRealloc  ########
209
210         .align  2
211         .globl  myRealloc
212         .ent    myRealloc
213 myRealloc:
214         .frame  $fp,44,ra
215         .set    noreorder
216         .cpload t9
217         .set    reorder
218
219         #Stack frame creation
220         subu    sp,sp,56
221
222         .cprestore 16
223         sw      ra,48(sp)
224         sw      $fp,44(sp)
225         sw      gp,40(sp)
226         move    $fp,sp
227
228         # Parameters
229         sw      a0,56($fp)  # Guardo en la direccion de memoria
230                             # 56($fp) la variable ptr (void * ptr).
231         sw      a1,60($fp)  # Guardo en la direccion de memoria 60(
                $fp)
232                             # la variable tamanyoNew (size_t
                                    tamanyoNew).
233         sw      a2,64($fp)  # Guardo en la direccion de memoria 64(
                $fp)
234                             # la variable tamanyoOld (int tamanyoOld
                                    ).
235
236         lw      v0,60($fp)  # Cargo en v0 el contenido de la
                variable tamanyoNew,
237                             # que esta en la direccion de memoria
                                    60($fp)
238         bne     v0,zero,$MyReallocContinueValidations # If (
                tamanyoNew != 0) goto MyReallocContinueValidations
239
240         # If (tamanyoNew == 0)
241         lw      a0,56($fp)  # Cargo en a0 la direccion de memoria
                guardada
242                             # en la direccion 56($fp), o sea, la
```

```
                                variable * ptr.
243         la      t9,myfree   # Cargo la direccion de la funcion
                myfree.
244         jal     ra,t9       # Ejecuto la funcion myfree.
245         sw      zero,56($fp)  # Coloco el puntero apuntando a NULL (
                ptr = NULL;).
246         sw      zero,32($fp)  # Coloco en la direccion de memoria
                32($fp) NULL,
247                               # que seria el resultado de la funcion
                                      myRealloc.
248         b       $MyReallocReturn # Salto incondicional para retornar
                 resultado de myRealloc.
249 $MyReallocContinueValidations:
250         lw      a0,60($fp)  # Cargo en a0 el contenido guardado en
                la direccion
251                               # 60($fp), o sea, la variable tamanyoNew
                                      .
252         la      t9,mymalloc  # Cargo la direccion de la funcion
                mymalloc.
253         jal     ra,t9        # Ejecuto la funcion mymalloc.
254         sw      v0,16($fp)  # Guardo en la direccion 16($fp) el
                contenido de v0, que
255                               # seria la direccion de la memoria
                                      asignada con mymalloc.
256         lw      v0,16($fp)  # Cargo en v0 la direccion de la memoria
                 asignada con
257                               # mymalloc (void * ptrNew = (void *)
                                      mymalloc(tamanyoNew);).
258
259         # (ptrNew == NULL)
260         # If (ptrNew != NULL) goto
                MyReallocContinueValidationsWithMemory
261         bne     v0,zero,$MyReallocContinueValidationsWithMemory
262         sw      zero,32($fp)  # Coloco en la direccion de memoria
                32($fp) NULL,
263                               # que seria el resultado de la funcion
                                      myRealloc.
264         b       $MyReallocReturn # Salto incondicional para retornar
                 resultado de myRealloc.
265 $MyReallocContinueValidationsWithMemory:
266         lw      v0,56($fp)  # Cargo en v0 la direccion de memoria
                guardada en la
267                               # direccion 56($fp), o sea, la variable
                                      * ptr.
268
269         # If (ptr != NULL) goto MyReallocContinueWithLoadCharacters
270         bne     v0,zero,$MyReallocContinueWithLoadCharacters
271
272         # (ptr == NULL)
273         lw      v0,16($fp)  # Cargo en v0 la direccion de memoria
                guardada en la
274                               # direccion 16($fp), o sea, la variable
                                      * ptrNew, que
275                               # seria la direccion de la memoria
                                      asignada con mymalloc.
276         sw      v0,32($fp)  # Coloco en la direccion de memoria 32(
                $fp) el contenido
277                               # de v0 (* ptrNew), que seria el
                                      resultado de la funcion myRealloc.
278         b       $MyReallocReturn # Salto incondicional para retornar
                 resultado de myRealloc.
279 $MyReallocContinueWithLoadCharacters:
280         lw      v0,60($fp)  # Cargo en v0 el contenido guardado en
                la direccion 60($fp),
281                               # o sea, la variable tamanyoNew.
282         sw      v0,20($fp)  # Guardo en la direccion de memoria 20(
                $fp) la variable
283                               # tamanyoNew guardada en v0 (int end =
                                      tamanyoNew;).
284
285         lw      v1,64($fp)  # Cargo en v1 el contenido guardado en
                la direccion
286                               # 64($fp), o sea, la variable tamanyoOld
                                      .
```

```
287         lw      v0,60($fp)  # Cargo en v0 el contenido guardado en
                la direccion
288                             # 60($fp), o sea, la variable tamanyoNew
                                    , para poder
289                             # luego hacer comparacion.
290
291         # (tamanyoOld < tamanyoNew)
292         sltu    v0,v1,v0  # Compara el contenido de la variable
                tamanyoOld (v1)
293                             # con tamanyoNew (v0), y guarda true en v0
                                    si el
294                             # primero (tamanyoOld) es mas chico que el
                                    segundo (tamanyoNew).
295         # If (tamanyoOld >= tamanyoNew) goto MyReallocLoadCharacters
296         beq     v0,zero,$MyReallocLoadCharacters # FALSE = 0
297         lw      v0,64($fp)  # Cargo en v0 el contenido guardado en
                la
298                             # direccion 64($fp), o sea, la variable
                                    tamanyoOld.
299         sw      v0,20($fp)  # Guardo en la direccion 20($fp), que
                seria la variable
300                             # end, el contenido de la variable
                                    tamanyoOld (end = tamanyoOld;).
301 $MyReallocLoadCharacters:
302         lw      v0,16($fp)  # Cargo en v0 el contenido guardado en
                la direccion 16($fp),
303                             # o sea, la variable ptrNew.
304         sw      v0,24($fp)  # Guardo en la direccion de memoria 24(
                $fp) el contenido de
305                             # v0 (char *tmp = ptrNew;).
306         lw      v0,56($fp)  # Cargo en v0 el contenido guardado en
                la direccion 56($fp),
307                             # o sea, la variable ptr.
308         sw      v0,28($fp)  # Guardo en la direccion de memoria 28(
                $fp) el contenido
309                             # de v0 (const char *src   = ptr;).
310 $MyReallocWhileLoadCharacter:
311         lw      v0,20($fp)  # Cargo en v0 el contenido guardado en
                la direccion 20($fp),
312                             # o sea, la variable end.
313         addu    v0,v0,-1    # Decremento en 1 el contenido de v0 (
                end --).
314         move    v1,v0       # Muevo el contenido de v0 a v1.
315         sw      v1,20($fp)  # Guardo en la direccion de memoria 20(
                $fp), que seria en donde
316                             # estaba end, el nuevo valor de end (
                                    habia sido decrementado en 1).
317         li      v0,-1       # Cargo en v0 el literal -1.
318         bne     v1,v0,$MyReallocContinueWhileLoad # If ( end != -1)
                goto MyReallocContinueWhileLoad.
319         b       $MyReallocFinalizedWhileLoad # Salto incondicional
                fuera del while, porque la variable end es -1.
320 $MyReallocContinueWhileLoad:
321         # *tmp = *src;
322         lw      v1,24($fp)  # Cargo en v1 el contenido guardado en
                la direccion 24($fp), que seria *tmp.
323         lw      v0,28($fp)  # Cargo en v0 el contenido guardado en
                la direccion 28($fp), que seria *src.
324         lbu     v0,0(v0)    # Cargo la direccion de memoria en v0 de
                 src.
325         sb      v0,0(v1)    # Guardo en la direccion apuntada por el
                 contenido de v1, la direccion de
326                             # memoria guardada en v0 (*tmp = *src;).
327
328         # tmp ++
329         lw      v0,24($fp)  # Cargo en v0 el contenido guardado en
                la direccion 24($fp), que seria *tmp.
330         addu    v0,v0,1     # Incremento en 1 el contenido guardado
                en v0 (tmp ++).
331         sw      v0,24($fp)  # Guardo en la direccion de memoria 24(
                $fp) lo que tenia v0
332                             # (el resultado de hacer tmp ++).
333
334         # src ++
```

```
335         lw      v0,28($fp)  # Cargo en v0 el contenido guardado en
                la direccion 28($fp), que seria *src.
336         addu    v0,v0,1     # Incremento en 1 el contenido guardado
                en v0 (src ++).
337         sw      v0,28($fp)  # Guardo en la direccion de memoria 28(
                $fp) lo que tenia v0
338                             # (el resultado de hacer src ++).
339
340         b       $MyReallocWhileLoadCharacter # Vuelvo a entrar al
                while
341 $MyReallocFinalizedWhileLoad:
342         lw      a0,56($fp)  # Cargo en v0 el contenido guardado en
                la direccion 56($fp), que seria *ptr.
343         la      t9,myfree   # Cargo la direccion de la funcion
                myfree.
344         jal     ra,t9       # Ejecuto la funcion myfree.
345         sw      zero,56($fp)  # Coloco el puntero apuntando a NULL (
                ptr = NULL;).
346
347         lw      v0,16($fp)  # Cargo en v0 la direccion de memoria
                guardada en la
348                             # direccion 16($fp), o sea, la variable
                                  * ptrNew, que seria
349                             # la direccion de la memoria asignada
                                  con mymalloc..
350         sw      v0,32($fp)  # Guardo en la direccion de memoria 32(
                $fp) el contenido de v0
351                             # (* ptrNew), que seria el resultado de
                                  la funcion myRealloc.
352 $MyReallocReturn:
353         lw      v0,32($fp)  # Cargo en v0 el resultado de la funcion
                 myRealloc guardado
354                             # en la direccion de memoria 32($fp).
355         move    sp,$fp
356         lw      ra,48(sp)
357         lw      $fp,44(sp)
358         addu    sp,sp,56
359         j       ra      # Jump and return
360
361         .end    myRealloc
```

Stack frame:

void * myRealloc(void * ptr, size_t tamanyoNew, int tamanyoOld)		
Offset	**Contents**	**Type reserved area**
64	tamanyoOld	**ABA (caller)**
60	tamanyoNew	
56	* ptr	
52	////////////////////////////////	**SRA**
48	ra	
44	fp	
40	gp	
36	////////////////////////////////	**LTA**
32	Resultado de la función	
28	* src	
24	* tmp	
20	end	
16	* ptrNew	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	
Stack frame: myRealloc		

Figura 17: Stack frame: memoryStackFrames

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 específicar 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. Asegurarse de tener los siguientes archivos:
tp1.c, bufferFunctions.S, palindromeFunctions.S, memoryFunctions.S.

```
$ gcc -Wall -o tp1 tp1.c *.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 obtuvieron los tiempo transcurridos entre distintas ejecuciones cambiando los parámetros 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. Adicionalmente se tomaron los tiempos con cronómetro para verificar que los tiempos arrojados por el comando time coincidas con los tomados por un instrumento físico distinto.

A continuación una tabla con los valores medidos:

Tamaño de archivo usado apróximadamente 834 kB.
Tamaño de línea en archivo apróximadamente: 1 byte * 450 char = 450 byte(caracteres/línea).

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.
Es de notar que un pequeño aumento en el tamaño del buffer(in/out) aumenta considerablemente el tiempo de ejecución del programa. Los tiempos tomados por cronómetro practicamente coinciden si se toma un error de medición de +-1s; teniendo en cuenta el tiempo de reacción.

id	stream input	stream output	real time[s]	user time[s]	sys time[s]	cron time[s]
1	1	1	60,02	4,99	37,79	60.95
2	2	2	51,14	4,01	30,00	51,38
4	5	5	32,77	2,87	22,75	33,22
5	10	10	27,10	2,78	20,00	27,38
6	50	50	21,00	2,62	17,05	21,39
7	100	100	19,43	2,53	16,24	19,77
8	300	300	18,90	2,54	16,16	19,10
9	600	600	18,35	2,41	15,64	18,58
10	1000	1000	17,95	2,43	15,30	18.31
11	2000	2000	17,93	2,29	15,49	18,14
12	3000	3000	18,02	2,16	15,64	18,39
13	5000	5000	17,70	2,42	15,14	18.06

Cuadro 1: Valores de la ejecución medidos con función time.

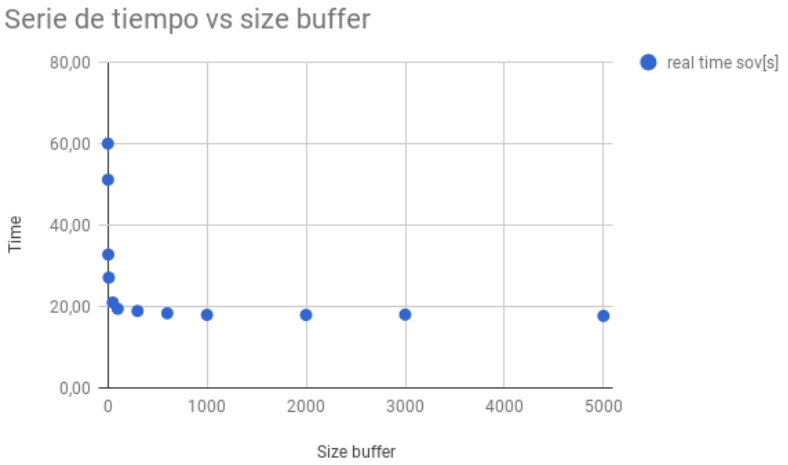

Figura 18: Gráfico de incidencia del buffer

Para tomar la medición a mano se uso un cronómetro electrónico de celular.

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. 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 disminuia 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