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:

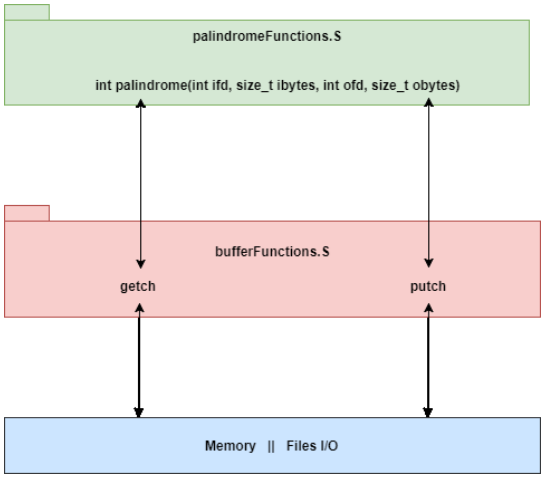


Figura 1: Se tiene dos grandes paquetes de funciones que hacen al proyecto, por un lado las asociadas a la funcionalidad de verificacion de léxicos palíndromos; y por otro lado, el encargado de proveer los caracteres que pueden ser parte de un léxico.

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   * Retorna un caracter (char) del ibuffer, y ser necesario
78   * lo recarga con los datos todavía no leídos del archivo de entrada
         .
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   * Si es palindromo, llama a putch para enviar el char
84   * al buffer. putch de todos los char que contiene el léxico.
85   */
86  int saveIfPalindrome() {
87          int itsPalindromic = verifyPalindromic();
88
89          if (itsPalindromic == TRUE) {
90                  int idx = 0;
91                  int error = FALSE;
92                  while(idx < lexico.quantityCharactersInBuffer &&
                         error == FALSE) {
93                          int result = putch(lexico.buffer[idx]);
94                          if (result == EOF) {
95                                  error = TRUE;
96                          }
97                          idx ++;
98                  }
99
100                 if (error == FALSE) {
101                         int result = putch('\n');
102                         if (result == EOF) {
103                                 error = TRUE;
104                         }
105                 }
106
107                 if (error == TRUE) {
108                         fprintf(stderr, "[Error] Error al escribir
                                 en el archivo output el palindromos. \n"
                                 );
109                         return ERROR_PUTCH;
110                 }
111         }
112         return OKEY;
113 }
114
115 /*
116  * Función principal.
117  * Si el palindromo es válido lo carga en el buffer.
118  */
119 int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes) {
120         initializeInput(ifd, ibytes);
121         initializeOutput(ofd, obytes);
122
123         lexico.quantityCharactersInBuffer = 0;
124         int icharacter = getch();
125         int result = OKEY;
126         while (icharacter != EOF && icharacter != ERROR_IREAD &&
                 result == OKEY) {
127                 char character = icharacter;
128
129                 if (isKeywords(character) == TRUE) {
130                         result = loadInBuffer(character, &lexico,
                                 LEXICO_BUFFER_SIZE);
131                 } else {
132                         // Dentro de esta funcion se invoca a putch
                                 si el lexico es palindromo.
```

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

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

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

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

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

```
232
233         # (bytesRead == 0)
234         lw      v0,32($fp)      # Cargo en v0 lo que esta en la
                direccion 32($fp), que seria bytesRead.
235         bne     v0,zero,loadBytesRead # If (bytesRead != 0) goto
                loadBytesRead.
236
237         # bytesRead is 0
238
239         # endIFile = TRUE;
240         li      v0,TRUE
241         sw      v0,endIFile
242 loadBytesRead:
243         # ibuffer.quantityCharactersInBuffer += bytesRead;
244         lw      v1,ibuffer+4    # Cargo en v1 ibuffer.
                quantityCharactersInBuffer (segundo
245                                 # elemento del struct Buffer).
246         lw      v0,32($fp)      # Cargo en v0 los bytes leidos (
                bytesRead).
247         addu    v0,v1,v0        # Guardo el resultado de la suma en
                v0:
248                                 # quantityCharactersInBuffer +
                                      bytesRead.
249         sw      v0,ibuffer+4    # Guardo el resultado de la suma en
                ibuffer.quantityCharactersInBuffer.
250
251         # bytesToRead = ibuffer.sizeBytes - ibuffer.
                quantityCharactersInBuffer;
252         lw      v1,ibuffer+8    # Cargo en v1 ibuffer.sizeBytes.
253         lw      v0,ibuffer+4    # Cargo en v0 ibuffer.
                quantityCharactersInBuffer.
254         subu    v0,v1,v0        # Guardo el resultado de la suma en
                v0:
255                                 # ibuffer.sizeBytes - ibuffer.
                                      quantityCharactersInBuffer
256         sw      v0,28($fp)      # Guardo el resultado de la suma en
                la direccion 28($fp), que
257                                 # representa la variable bytesToRead
                                      .
258
259         # (bytesToRead <= 0)
260         lw      v0,28($fp)      # Cargo en v0 bytesToRead.
261         bgtz    v0,whileLoadIBuffer # If (bytesToRead > 0) goto
                whileLoadIBuffer
262
263         # bytesToRead is <= 0
264
265         # completeDelivery = TRUE;
266         li      v0,TRUE
267         sw      v0,24($fp)      # Guardo TRUE en la direccion 24($fp
                ), que
268                                 # representa la variable
                                      completeDelivery.
269         b       whileLoadIBuffer # Vuelvo a intentar entrar al loop.
270 initializeLastPositionInIBufferRead:
271         # lastPositionInIBufferRead = -1;
272         li      v0,-1
273         sw      v0,lastPositionInIBufferRead
274
275         # return OKEY_I_FILE;
276         li      v0,OKEY_I_FILE
277         sw      v0,36($fp)
278 returnLoadIBufferWithFile:
279         lw      v0,36($fp)
280         move    sp,$fp
281         lw      ra,48(sp)
282         lw      $fp,44(sp)
283         # destruyo stack frame
284         addu    sp,sp,56
285         # vuelvo a funcion llamante
286         j       ra
287
288         .end    loadIBufferWithIFile
289
```

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

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

```
417
418         # (obuffer.buffer == NULL)
419         lw      v0,obuffer      # Cargo en v0 obuffer.buffer (primer
                elemento del
420                                 # struct Buffer.
421         beq     v0,zero,returnOkey # If (obuffer.buffer == NULL)
                goto returnOkey
422
423         # obuffer.buffer is not NULL
424
425         # (obuffer.quantityCharactersInBuffer <= 0)
426         lw      v0,obuffer+4    # Cargo en v0 obuffer.
                quantityCharactersInBuffer
427                                 # (segundo elemento del struct
                                      Buffer).
428         blez    v0,returnOkey   # If (obuffer.
                quantityCharactersInBuffer <= 0)
429                                 # goto returnOkey.
430         b       loadOBuffer     # Salto incondicional a loadOBuffer.
431 returnOkey:
432         # return OKEY;
433         sw      zero,40($fp)    # Guardo en la direccion 40($fp) el
                resultado de la
434                                 # funcion, en este caso OKEY (= 0).
435         b       returnWriteBufferInOFile # Salto incondicional al
                return de la funcion.
436 loadOBuffer:
437         # int completeDelivery = FALSE (= 0)
438         sw      zero,24($fp)    # 24($fp) <-> completeDelivery
439
440         # int bytesWriteAcum = 0;
441         sw      zero,28($fp)    # 28($fp) <-> bytesWriteAcum
442
443         # int bytesToWrite = obuffer.quantityCharactersInBuffer;
444         lw      v0,obuffer+4    # quantityCharactersInBuffer es el
                segundo elemento del
445                                 # struct Buffer.
446         sw      v0,32($fp)      # 32($fp) <-> bytesToWrite
447 whileWriteOFile:
448         # (completeDelivery == FALSE)
449         lw      v0,24($fp)      # Cargo en v0 completeDelivery
450         beq     v0,FALSE,inWhileWriteOFile  # If (completeDelivery
                == FALSE) goto inWhileWriteOFile
451
452         # completeDelivery is not FALSE (is TRUE)
453         b       loadReturnOkey # Salto incondicional a
                loadReturnOkey.
454 inWhileWriteOFile:
455         # int bytesWrite = write(ofd, obuffer.buffer +
                bytesWriteAcum, bytesToWrite);
456
457         # obuffer.buffer + bytesWriteAcum
458         lw      v1,obuffer      # obuffer.buffer es el primer
                elemento del struct Buffer.
459         lw      v0,28($fp)      # Cargo en v0 bytesWriteAcum.
460         addu    v0,v1,v0        # Me muevo por buffer, guardo la
                direccion en v0.
461
462         lw      a0,ofd          # Cargo en a0 ofd. Parametro de la
                funcion write.
463         move    a1,v0           # Cargo en a1 la direccion sobre
                obuffer.buffer.
464                                 # Parametro de la funcion write.
465         lw      a2,32($fp)      # Cargo en a2 bytesToWrite. Parametro
                 de la funcion write.
466
467         li      v0, SYS_write
468         syscall  # Seria int bytesWrite = write(ofd, obuffer.buffer
                + bytesWriteAcum, bytesToWrite);
469
470         beq     a3,zero,saveBytesWrite # Si no hubo error, salto a
                saveBytesWrite.
471
472         # Hubo error al querer escribir en el archivo => Mensaje de
```

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

```
528 putch:
529         .frame  $fp,48,ra
530         .set    noreorder
531         .cpload t9
532         .set    reorder
533 
534         # Stack frame creation
535         subu    sp,sp,48
536 
537        .cprestore 16
538         sw      ra,40(sp)
539         sw      $fp,36(sp)
540         sw      gp,32(sp)
541 
542         # de aqui al fin de la funcion uso $fp en lugar de sp.
543         move    $fp,sp
544 
545         # Parametro
546         sw      a0,48($fp)      # Guardo en la direccion de memoria
                48($fp) lo que tiene a0, que
547                                 # es el parametro que recibe la
                                        funcion, int character.
548 
549         # (obuffer.buffer == NULL)
550         lw      v0,obuffer      # Cargo en v0 obuffer.buffer, que es
                 el primer elemento del
551                                 # struct Buffer.
552         bne     v0,zero,loadInOBuffer # If (obuffer.buffer != NULL)
                goto loadInOBuffer
553 
554         # obuffer.buffer is NULL
555 
556         # Asigno memoria a obuffer.buffer
557         # obuffer.buffer = (char *) malloc(obuffer.sizeBytes*sizeof(
                char));
558         lw      a0,obuffer+8    # quantityCharactersInBuffer es el
                tercer elemento del struct Buffer.
559         la      t9,mymalloc
560         jal     ra,t9
561         sw      v0,obuffer
562         lw      v0,obuffer
563 
564         # Verifico error en la asignacion de memoria.
565         # (obuffer.buffer == NULL)
566         bne     v0,zero,initializeQuantityCharactersInOBuffer
567         # If (obuffer.buffer != NULL) goto
568         # initializeQuantityCharactersInOBuffer.
569 
570         # obuffer.buffer is NULL => Mensaje de error.
571         li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
572         la      a1,MENSAJE_ERROR_MEMORIA_OBUFFER # Cargo en a1 la
                direccion de memoria donde se encuentra el mensaje a
                cargar.
573         li      a2,BYTES_MENSAJE_ERROR_MEMORIA_OBUFFER  # Cargo en
                a2 la cantidad de bytes a escribir.
574         li      v0, SYS_write
575         syscall    # No controlo error porque sale de por si de la
                funcion por error.
576 
577         # return ERROR_MEMORY;
578         li      v0,ERROR_MEMORY
579         sw      v0,24($fp)      # Cargo en la direccion 24($fp) el
                resultado de la funcion,
580                                 # en este caso es el codigo de error
                                        ERROR_MEMORY.
581         b       returnPutch     # Salto incondicional a returnPutch.
582 initializeQuantityCharactersInOBuffer:
583         # obuffer.quantityCharactersInBuffer = 0;
584         sw      zero,obuffer+4
585 loadInOBuffer:
586         # obuffer.buffer[obuffer.quantityCharactersInBuffer] =
                character;
587         lw      v1,obuffer      # Cargo obuffer.buffer en v1.
```

```
588         lw      v0,obuffer+4    # Cargo obuffer.
                quantityCharactersInBuffer en v0.
589         addu    v1,v1,v0        # Me muevo sobre obuffer.buffer, o
                sea:
590                                 # obuffer.buffer[obuffer.
                                        quantityCharactersInBuffer]
591                                 # Guardo esta direccion en v1.
592         lbu     v0,48($fp)      # Cargo en v0 character.
593         sb      v0,0(v1)        # Asigno character a obuffer.buffer[
                obuffer.quantityCharactersInBuffer].
594
595         # obuffer.quantityCharactersInBuffer ++;
596         lw      v0,obuffer+4
597         addu    v0,v0,1
598         sw      v0,obuffer+4
599
600         # (obuffer.quantityCharactersInBuffer == obuffer.sizeBytes)

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

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

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

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

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

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

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

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

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: initializeInput

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: getch

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: freeResources

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: flush

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: cleanContentBuffer

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: loadIBufferWithIFile

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: loadInBuffer

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: putch

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: writeBufferInOFile

4.2. Código MIPS32: palindromeFunctions.S

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #include "constants.h"
5
6  # Size mensajes
7  #define BYTES_MENSAJE_ERROR_PUTCH    65
8
9
10 #------ toLowerCase ------##
11
12         .text
13         .align  2
14         .globl  toLowerCase
15         .ent    toLowerCase
16 toLowerCase:
17         .frame  $fp,24,ra
18         .set    noreorder
19         .cpload t9
20         .set    reorder
21
22         # Stack frame creation
23         subu    sp,sp,24
24
25         .cprestore 0
26         sw      $fp,20(sp)
27         sw      gp,16(sp)
28
29         # de aqui al fin de la funcion uso $fp en lugar de sp.
30         move    $fp,sp
31
32         # Parametro
33         sb      a0,8($fp)       # Guardo en la direccion 8($fp) el
               contenido de
34                                 # a0 que es word (char word).
35
36         # (word >= 65 && word <= 90)
37
38         # (word >= 65)
39         lb      v0,8($fp)
40         slt     v0,v0,65        # Guardo en v0 TRUE si word es mas
               chico que 65, sino guardo FALSE.
41         bne     v0,FALSE,returnToLowerCase  # If (word < 65) goto
               returnToLowerCase
42
43         # word is >= 65
44
45         # (word <= 90)
46         lb      v0,8($fp)
47         slt     v0,v0,91        # Guardo en v0 TRUE si word es mas
               chico que 91, sino guardo FALSE.
48         beq     v0,FALSE,returnToLowerCase # If (word >= 91) goto
               returnToLowerCase
49
50         # word is <= 90
51
52         # word += 32;
53         lbu     v0,8($fp)
54         addu    v0,v0,32
55         sb      v0,8($fp)  # Guardo en la direccion 8($fp) el
               resultado de la funcion,
56                            # que coincide con la variable word.
57 returnToLowerCase:
58         lb      v0,8($fp)
59         move    sp,$fp
60         lw      $fp,20(sp)
61         # destruyo stack frame
62         addu    sp,sp,24
63         # vuelvo a funcion llamante
64         j       ra
```

```
65
66          .end    toLowerCase
67
68
69
70  ##------ verifyPalindromic ------##
71
72          .align  2
73          .globl  verifyPalindromic
74          .ent    verifyPalindromic
75  verifyPalindromic:
76          .frame  $fp,72,ra
77          .set    noreorder
78          .cpload t9
79          .set    reorder
80
81          subu    sp,sp,72
82
83          .cprestore 16
84          sw      ra,64(sp)
85          sw      $fp,60(sp)
86          sw      gp,56(sp)
87
88          move    $fp,sp
89
90          # (lexico.buffer == NULL || lexico.
                quantityCharactersInBuffer <= 0)
91
92          # (lexico.buffer == NULL)
93          lw      v0,lexico
94          beq     v0,zero,returnFalse # If (lexico.buffer == NULL)
                goto returnFalse
95
96          # lexico.buffer is not NULL
97
98          # (lexico.quantityCharactersInBuffer <= 0)
99          lw      v0,lexico+4
100         blez    v0,returnFalse   # If (lexico.
                quantityCharactersInBuffer <= 0) returnFalse
101
102         # lexico.quantityCharactersInBuffer is > 0
103         b       verifyOneCharacter
104 returnFalse:
105         # return FALSE;
106         sw      zero,48($fp)     # FALSE = 0, guardo en la direccion
                 48($fp) el resultado de la funcion
107         b       returnVerifyPalindromic
108 verifyOneCharacter:
109         # (lexico.quantityCharactersInBuffer == 1)
110         lw      v1,lexico+4
111         li      v0,1
112         bne     v1,v0,compareCharacters # IF (lexico.
                quantityCharactersInBuffer != 1)
113                                         # goto compareCharacters
114
115         # lexico.quantityCharactersInBuffer is equals to 1
116
117         # return TRUE;
118         li      v0,TRUE
119         sw      v0,48($fp)    # guardo en la direccion 48($fp) el
                resultado de la funcion
120         b       returnVerifyPalindromic
121 compareCharacters:
122         # double middle = (double)lexico.quantityCharactersInBuffer
                / 2;
123         l.s     $f0,lexico+4  # Cargo lexico.
                quantityCharactersInBuffer en f0
124         cvt.d.w $f2,$f0       # Convierto el integer
                quantityCharactersInBuffer a double
125         l.d     $f0,doubleWord   # Cargo en f0 el valor 2.
126         div.d   $f0,$f2,$f0     # Division con Double (double)
                quantityCharacterInWord / 2;
127                                 # Sintaxis: div.d FRdest, FRsrc1,
                                        FRsrc2
```

```
128         s.d     $f0,24($fp)     # Guarda el resultado de la division
                en 24($fp). O sea,
129                                 # middle (double middle = (double)
                                        quantityCharacterInWord / 2;)
130
131         # int idx = 0;
132         sw      zero,32($fp)    # En 32($fp) se encuentra idx (int
                idx = 0;).
133
134         # int validPalindromic = TRUE;
135         li      v0,TRUE
136         sw      v0,36($fp)    # En 36($fp) esta la variable
                validPalindromic.
137
138         # int last = lexico.quantityCharactersInBuffer - 1;
139         lw      v0,lexico+4
140         addu    v0,v0,-1
141         sw      v0,40($fp)    # En 40($fp) esta la variable last.
142 whileMirror:
143         # (idx < middle && last >= middle && validPalindromic ==
                TRUE)
144         l.s     $f0,32($fp)   # Cargo idx en f0.
145         cvt.d.w $f2,$f0       # Convierto el integer idx a double y
                lo guardo en
146                               # f2 para poder hacer la comparacion.
147         l.d     $f0,24($fp)   # Cargo en a0 la variable middle.
148         c.lt.d  $f2,$f0       # Compara la variable idx con la
                variable middle, y
149                               # setea el condition flag en true si
                                      el primero (idx) es
150                               # mas chico que el segundo (middle).
151         bc1t    verifyConditionLastWithMiddle # Si el condition flag
                es true, continua
152                                                 # haciendo las
                                                        comparaciones.
153         b       whileMirrorFinalized   # Si el condition flag es
                false, salta al final de la
154                                        # funcion, devolviendo el
                                               valor de la variable
                                               validPalindromic.
155 verifyConditionLastWithMiddle:
156         l.s     $f0,40($fp)  # Cargo la variable last en f0.
157         cvt.d.w $f2,$f0      # Convierto el integer last a double y
                lo guardo
158                              # en f2 para poder hacer la comparacion
                                       .
159         l.d     $f0,24($fp)  # Cargo en f0 el contenido de la
                variable middle.
160         c.le.d  $f0,$f2      # Compara el contenido de la variable
                last con la variable
161                              # middle, y setea el condition flag en
                                       true si
162                              # middle es menor o igual a last, sino
                                       false
163         bc1t    verifyConditionPalindromicTrue # Si el condition
                flag es true,
164                                                # continua haciendo
                                                       las comparaciones
                                                       .
165         b       whileMirrorFinalized  # false
166 verifyConditionPalindromicTrue:
167         lw      v1,36($fp)  # Carga en v1 validPalindromic
168         li      v0,TRUE
169         beq     v1,v0,whileMirrorContent  # If (validPalindromic ==
                TRUE) goto whileMirrorContent
170         b       whileMirrorFinalized
171 whileMirrorContent:
172         # char firstCharacter = toLowerCase(lexico.buffer[idx]);
173         lw      v1,lexico
174         lw      v0,32($fp)
175         addu    v0,v1,v0
176         lb      v0,0(v0)
177         move    a0,v0
178         la      t9,toLowerCase
```

```
179         jal     ra,t9
180         sb      v0,44($fp)   # Guardo en la direccion 44($fp) la
                variable firstCharacter
181
182         # char lastCharacter = toLowerCase(lexico.buffer[last]);
183         lw      v1,lexico
184         lw      v0,40($fp)
185         addu    v0,v1,v0
186         lb      v0,0(v0)
187         move    a0,v0
188         la      t9,toLowerCase
189         jal     ra,t9
190         sb      v0,45($fp)   # Guardo en la direccion 45($fp) la
                variable lastCharacter
191
192         # (firstCharacter != lastCharacter)
193         lb      v1,44($fp)   # Cargo firstCharacter en v1
194         lb      v0,45($fp)   # Cargo lastCharacter en v0
195         beq     v1,v0,continueWhile # If (firstCharacter ==
                lastCharacter) goto continueWhile
196
197         # firstCharacter != lastCharacter
198
199         # validPalindromic = FALSE;
200         sw      zero,36($fp)
201 continueWhile:
202         # idx ++;
203         lw      v0,32($fp)
204         addu    v0,v0,1
205         sw      v0,32($fp)
206
207         # last --;
208         lw      v0,40($fp)
209         addu    v0,v0,-1
210         sw      v0,40($fp)
211
212         b       whileMirror
213 whileMirrorFinalized:
214         # return validPalindromic;
215         lw      v0,36($fp)   # Cargo el contenido de
                validPalindromic en v0
216         sw      v0,48($fp)   # Guardo en la direccion 48($fp) el
                resultado de la funcion
217                              # que esta en v0.
218 returnVerifyPalindromic:
219         lw      v0,48($fp)
220         move    sp,$fp
221         lw      ra,64(sp)
222         lw      $fp,60(sp)
223         # destruyo stack frame
224         addu    sp,sp,72
225         # vuelvo a funcion llamante
226         j       ra
227
228         .end    verifyPalindromic
229
230
231
232 ###------ isKeywords ------##
233
234         .align  2
235         .globl  isKeywords
236         .ent    isKeywords
237 isKeywords:
238         .frame  $fp,24,ra
239         .set    noreorder
240         .cpload t9
241         .set    reorder
242
243         # Stack frame creation
244         subu    sp,sp,24
245
246         .cprestore 0
247         sw      $fp,20(sp)
```

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

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

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

```
432         sw      v0,32($fp)
433 returnWithError:
434         # (error == TRUE)
435         lw      v1,32($fp)
436         li      v0,TRUE
437         bne     v1,v0,returnOkeySaveIfPalindrome # If (error != TRUE
                ) goto returnOkeySaveIfPalindrome
438
439         # Mensaje de error
440         li      a0,FILE_DESCRIPTOR_STDERR # Cargo en a0
                FILE_DESCRIPTOR_STDERR.
441         la      a1,MENSAJE_ERROR_PUTCH  # Cargo en a1 la direccion
                de memoria donde se encuentra el mensaje a cargar.
442         li      a2,BYTES_MENSAJE_ERROR_PUTCH    # Cargo en a2 la
                cantidad de bytes a escribir.
443         li      v0, SYS_write
444         syscall    # No controlo error porque sale de por si de la
                funcion por error.
445
446         # return ERROR_PUTCH;
447         li      v0,ERROR_PUTCH
448         sw      v0,40($fp)
449         b       returnSaveIfPalindrome
450 returnOkeySaveIfPalindrome:
451         sw      zero,40($fp)    # OKEY = 0
452 returnSaveIfPalindrome:
453         lw      v0,40($fp)
454         move    sp,$fp
455         lw      ra,56(sp)
456         lw      $fp,52(sp)
457         # destruyo stack frame
458         addu    sp,sp,64
459         # vuelvo a funcion llamante
460         j       ra
461
462         .end    saveIfPalindrome
463
464
465
466 #######  palindrome  ########
467
468         .align  2
469         .globl  palindrome
470         .ent    palindrome
471 palindrome:
472         .frame  $fp,56,ra
473         .set    noreorder
474         .cpload t9
475         .set    reorder
476
477         # Stack frame creation
478         subu    sp,sp,56
479
480         .cprestore 16
481         sw      ra,48(sp)
482         sw      $fp,44(sp)
483         sw      gp,40(sp)
484
485         # de aqui al fin de la funcion uso $fp en lugar de sp.
486         move    $fp,sp
487
488         # Parametros
489         sw      a0,56($fp)      # Guardo en la direccion 56($fp) ifd
                que estaba en a0.
490         sw      a1,60($fp)      # Guardo en la direccion 60($fp)
                ibytes que estaba en a1.
491         sw      a2,64($fp)      # Guardo en la direccion 64($fp) ofd
                que estaba en a2.
492         sw      a3,68($fp)      # Guardo en la direccion 68($fp)
                obytes que estaba en a3.
493
494         # initializeInput(ifd, ibytes);
495         lw      a0,56($fp)
496         lw      a1,60($fp)
```

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

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

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

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: isKeywords

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: varifyPalindromic

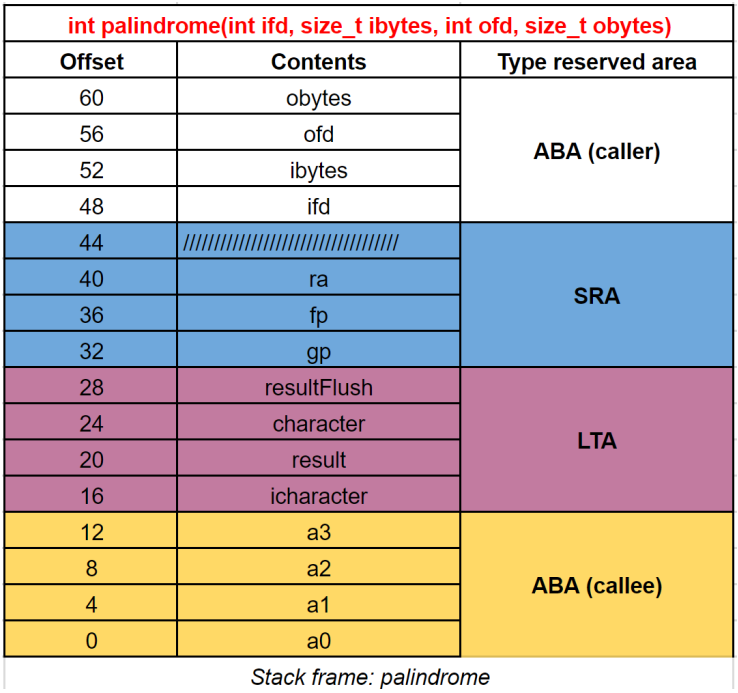
Stack frame:

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: toLowerCase

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: palindromo

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.

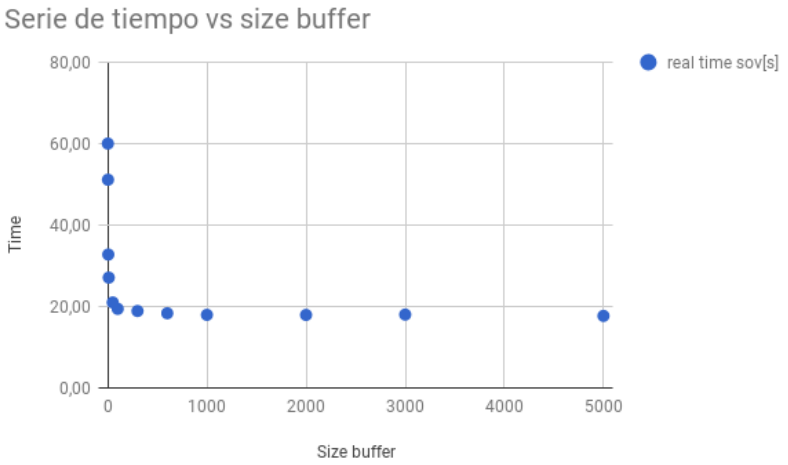


Figura 18: Gráfico de incidencia del buffer

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