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.
  - Sin especificación del archivo de entrada ni de salida.
- Setting del tamaño del buffer in y buffer out; indicando de a cuantos caracteres se debe leer y escribir.

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

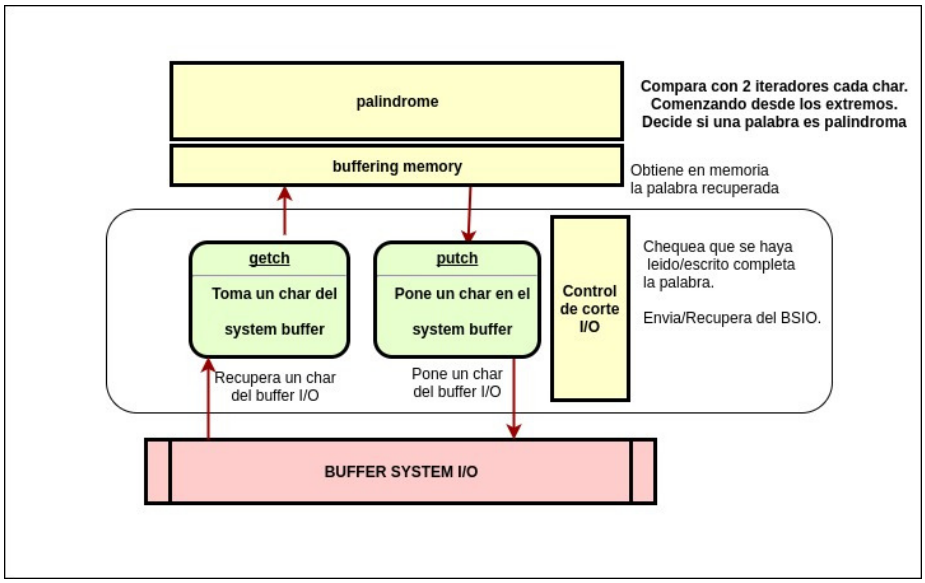

Figura 1: Diagrama: disposición palindrome

3. Implementación

3.1. Código fuente en lenguaje C: bufferFunctions.c

```
1  /*
2   * bufferFunctions.c
3   *
4   */
5
```

```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;

                if (bytesToRead <= 0) {
                        completeDelivery = TRUE;
                }
        }

        lastPositionInIBufferRead = -1;

        return OKEY_I_FILE;
}
```

```
75
76  /*
77   * Obtengo un caracter(char) del input file.
78   * Lo seteo en el buffer.
79   */
80  int getch() {
81          if (ibuffer.buffer == NULL || lastPositionInIBufferRead == (
                ibuffer.quantityCharactersInBuffer - 1)) {
82                  if (endIFile == TRUE) {
83                          return EOF;
84                  }
85                  int resultLoadIBuffer = loadIBufferWithIFile();
86                  if (resultLoadIBuffer == ERROR_I_READ) {
87                          return ERROR_I_READ;
88                  }
89
90                  if (ibuffer.quantityCharactersInBuffer == 0) {
91                          return EOF;
92                  }
93          }
94
95          lastPositionInIBufferRead ++;
96          return ibuffer.buffer[lastPositionInIBufferRead];
97  }
98
99  /*
100  * Escribe los caracteres en el output file
101  * de acuerdo al tamaño del buffer.
102  */
103 int writeBufferInOFile() {
104         if (obuffer.buffer == NULL || obuffer.
                quantityCharactersInBuffer <= 0) {
105                 return OKEY;
106         }
107
108         int completeDelivery = FALSE;
109         int bytesWriteAcum = 0;
110         int bytesToWrite = obuffer.quantityCharactersInBuffer;
111         while (completeDelivery == FALSE) {
112                 int bytesWrite = write(ofd, obuffer.buffer +
                        bytesWriteAcum, bytesToWrite);
113                 if (bytesWrite < 0) {
114                         fprintf(stderr, "[Error] Hubo un error al
                                escribir en el archivo. \n");
115                         return ERROR_WRITE;
116                 }
117
118                 bytesWriteAcum += bytesWrite;
119                 bytesToWrite = obuffer.quantityCharactersInBuffer -
                        bytesWriteAcum;
120
121                 if (bytesToWrite <= 0) {
122                         completeDelivery = TRUE;
123                 }
124         }
125
126         return OKEY;
127 }
128
129 /*
130  * Coloca un char en el output buffer.
131  * Llama a la escritura en el output file
132  * de ser necesario.
133  */
134 int putch(int character) {
135         if (obuffer.buffer == NULL) {
136                 obuffer.buffer = (char *) malloc(obuffer.sizeBytes*
                        sizeof(char));
137                 if (obuffer.buffer == NULL) {
138                         fprintf(stderr, "[Error] Hubo un error de
                                asignacion de memoria (obuffer). \n");
139                         return ERROR_MEMORY;
140                 }
141
```

```
142                     obuffer.quantityCharactersInBuffer = 0;
143             }
144
145             obuffer.buffer[obuffer.quantityCharactersInBuffer] =
                    character;
146             obuffer.quantityCharactersInBuffer ++;
147
148             if (obuffer.quantityCharactersInBuffer == obuffer.sizeBytes)
                    {
149                     writeBufferInOFile();
150                     obuffer.quantityCharactersInBuffer = 0;
151             }
152
153             return OKEY;
154     }
155     /*
156      * Flusea el contenido del buffer
157      */
158     int flush() {
159             if (obuffer.buffer != NULL && obuffer.
                    quantityCharactersInBuffer > 0) {
160                     return writeBufferInOFile();
161             }
162
163             return OKEY;
164     }
165
166     /*
167      * Libera los recursos solicitados por ibuffer/obuffer.
168      */
169     void freeResources() {
170             if (ibuffer.buffer != NULL) {
171                     free(ibuffer.buffer);
172                     ibuffer.buffer = NULL;
173             }
174
175             if (obuffer.buffer != NULL) {
176                     free(obuffer.buffer);
177                     obuffer.buffer = NULL;
178             }
179     }
180
181     /*
182      * Carga el caracter en el buffer
183      */
184     int loadInBuffer(char character, Buffer * buffer, size_t sizeInitial
            ) {
185             if (buffer->buffer == NULL) {
186                     buffer->buffer = malloc(sizeInitial * sizeof(char));
187                     buffer->sizeBytes = sizeInitial;
188             } else if (buffer->quantityCharactersInBuffer >= buffer->
                    sizeBytes) {
189                     size_t bytesLexicoPreview = buffer->sizeBytes;
190                     //Se hace una reasignacion exponencial del espacio.
191                     buffer->sizeBytes = bytesLexicoPreview * 2;
192                     // Esto es para no perder memoria.
193                     char * auxiliary = myRealloc(buffer->buffer, buffer
                            ->sizeBytes*sizeof(char), bytesLexicoPreview);
194                     if (auxiliary == NULL) {
195                             cleanContentBuffer(buffer);
196                     } else {
197                             buffer->buffer = auxiliary;
198                     }
199             }
200
201             if (buffer->buffer == NULL) {
202                     fprintf(stderr, "[Error] Hubo un error en memoria (
                            lexico). \n");
203                     return ERROR_MEMORY;
204             }
205
206             buffer->buffer[buffer->quantityCharactersInBuffer] =
                    character;
207             buffer->quantityCharactersInBuffer ++;
```

```
208
209         return OKEY;
210 }
211
212 /*
213  * Limpia el contenido del buffer pasado por parámetro.
214  */
215 void cleanContentBuffer(Buffer * buffer) {
216         if (buffer->buffer != NULL) {
217                 free(buffer->buffer);
218                 buffer->buffer = NULL;
219         }
220
221         buffer->quantityCharactersInBuffer = 0;
222         buffer->sizeBytes = 0;
223 }
```

3.2. 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 }
```

3.3. Código fuente en lenguaje C: tp1.c

```
1  /*
2   ============================================================================
3   Name        : tp1.c
4   Author      : Grupo orga 66.20
5   Version     : 1
```

```
6   Copyright   : Orga6620 - Tp1
7   Description : Trabajo practico 1: Programacion MIPS
8   ================================================================================
9   */
10
11  #include <stdio.h>
12  #include <stdlib.h>
13  #include <string.h>
14  #include <getopt.h>
15  #include <unistd.h>
16
17  #include "constants.h"
18  #include "palindromeFunctions.h"
19
20  #define VERSION "1.2"
21
22  size_t isize = 1;
23  size_t osize = 1;
24
25  int executeHelp() {
26          fprintf(stdout, "Usage: \n");
27          fprintf(stdout, "       tp1 -h \n");
28          fprintf(stdout, "       tp1 -V \n");
29          fprintf(stdout, "       tp1 [options] \n");
30          fprintf(stdout, "Options: \n");
31          fprintf(stdout, "       -V, --version           Print
                version and quit. \n");
32          fprintf(stdout, "       -h, --help              Print this
                information. \n");
33          fprintf(stdout, "       -i, --input             Location of
                the input file. \n");
34          fprintf(stdout, "       -o, --output            Location of
                the output file. \n");
35          fprintf(stdout, "       -I, --ibuf-bytes        Byte-count
                of the input buffer. \n");
36          fprintf(stdout, "       -O, --obuf-bytes        Byte-count
                of the output buffer. \n");
37          fprintf(stdout, "Examples: \n");
38          fprintf(stdout, "       tp1 -i ~/input -o ~/output \n");
39
40          return OKEY;
41  }
42
43  int executeVersion() {
44          fprintf(stdout, "Version: \"%s\" \n", VERSION);
45
46          return OKEY;
47  }
48
49  int executeByMenu(int argc, char **argv) {
50          int inputFileDefault = FALSE;
51          int outputFileDefault = FALSE;
52          FILE * fileInput = stdin;
53          FILE * fileOutput = stdout;
54
55          // Always begins with /
56          if (argc == 1) {
57                  // Run with default parameters
58                  inputFileDefault = TRUE;
59                  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 }
```

3.4. Código fuente en lenguaje C: palindromeFunctions.c

```
1  /*
2   * palindromeFunctions.c
3   *
4   */
5
6  #include "palindromeFunctions.h"
7
8  /*
9   * Contiene la palabra leida.
10  */
11 Buffer lexico;
12
13 /*
14  * Los caracteres válidos son aquellos que
15  * se encuentran dentro del rango ASCII:
16  * A - Z = [65 - 90]
17  * a - z = [97 - 122]
18  * 0 - 9 = [48 - 57]
19  * - =  45
20  * _ = 95
21  */
22 char toLowerCase(char word) {
23         /* ASCII:
24          *              A - Z = [65 - 90]
25          *              a - z = [97 - 122]
26          *              0 - 9 = [48 - 57]
27          *              - =  45
28          *              _ = 95
29          */
30         if (word >= 65 && word <= 90) {
31             word += 32;
32         }
33
```

```
34          return word;
35  }
36
37  /*
38   * Pre: lexico esta seteado. Lexico contiene la palabra.
39   * Verifica que el lexico(palabra) sea palindroma.
40   */
41  int verifyPalindromic() {
42          if (lexico.buffer == NULL || lexico.
                quantityCharactersInBuffer <= 0) {
43                  return FALSE;
44          }
45
46          /*
47           * Las palabras de 1 sólo caracter válido
48           * son siempre palindromas.
49           */
50          if (lexico.quantityCharactersInBuffer == 1) {
51                  // The word has one character
52                  return TRUE;
53          }
54
55          double middle = (double)lexico.quantityCharactersInBuffer /
                2;
56          int idx = 0;
57          int validPalindromic = TRUE;
58          int last = lexico.quantityCharactersInBuffer - 1;
59          while(idx < middle && last >= middle && validPalindromic ==
                TRUE) {
60                  char firstCharacter = toLowerCase(lexico.buffer[idx
                        ]);
61                  char lastCharacter = toLowerCase(lexico.buffer[last
                        ]);
62                  if (firstCharacter != lastCharacter) {
63                          validPalindromic = FALSE;
64                  }
65
66                  idx ++;
67                  last --;
68          }
69
70          return validPalindromic;
71  }
72
73  /*
74   * Verifica si un determinado caracter es un
75   * 'caracter válido' para procesar.
76   */
77  int isKeywords(char character) {
78          /* ASCII:
79           *              A - Z = [65 - 90]
80           *              a - z = [97 - 122]
81           *              0 - 9 = [48 - 57]
82           *              - =  45
83           *              _ = 95
84           */
85          if ((character >= 65 && character <= 90) || (character >= 97
                && character <= 122)
86                          || (character >= 48 && character <= 57)
87                          || character == 45 || character == 95) {
88                  return TRUE;
89          }
90
91          return FALSE;
92  }
93
94  /*
95   * Verifica si es palindromo.
96   * Si es palindromo, llama a putch para enviar el char
97   * al buffer.
98   */
99  int saveIfPalindrome() {
100         int itsPalindromic = verifyPalindromic();
101
```

```
102         if (itsPalindromic == TRUE) {
103                 int idx = 0;
104                 int error = FALSE;
105                 while(idx < lexico.quantityCharactersInBuffer &&
                        error == FALSE) {
106                         int result = putch(lexico.buffer[idx]);
107                         if (result == EOF) {
108                                 error = TRUE;
109                         }
110                         idx ++;
111                 }
112
113                 if (error == FALSE) {
114                         int result = putch('\n');
115                         if (result == EOF) {
116                                 error = TRUE;
117                         }
118                 }
119
120                 if (error == TRUE) {
121                         fprintf(stderr, "[Error] Error al escribir
                                en el archivo output la palabra %",
                                lexico.buffer);
122                         return ERROR_PUTCH;
123                 }
124         }
125         return OKEY;
126 }
127
128 /*
129  * Función principal. Verifica si un caracter(char) es válido.
130  * Si el char es válido lo carga en el buffer.
131  */
132 int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes) {
133         initializeInput(ifd, ibytes);
134         initializeOutput(ofd, obytes);
135
136         lexico.quantityCharactersInBuffer = 0;
137         int icharacter = getch();
138         int result = OKEY;
139         while (icharacter != EOF && icharacter != ERROR_I_READ &&
                result == OKEY) {
140                 char character = icharacter;
141
142                 if (isKeywords(character) == TRUE) {
143                         result = loadInBuffer(character, &lexico,
                                LEXICO_BUFFER_SIZE);
144                 } else {
145                         //Si el caracter NO es válido -> se deberia
                                dejar de procesar!
146                         //
147                         // Dentro de esta funcion se invoca a putch
                                si el lexico es palindromo.
148                         result = saveIfPalindrome();
149
150                         cleanContentBuffer(&lexico);
151                 }
152
153                 icharacter = getch();
154         }
155
156         // Guardo lo que haya quedado en lexico si es palindromo.
157         int resultFlush = saveIfPalindrome();
158         if (result == OKEY) {
159                 result = resultFlush;
160         }
161
162         cleanContentBuffer(&lexico);
163
164         resultFlush = flush();
165         if (result == OKEY) {
166                 result = resultFlush;
167         }
168         freeResources();
```

```
        return result;
}
```

4. Código MIPS32

4.1. Código MIPS32: bufferFunctions.S

```
CODIGO ACA
```

include ¡mips/regdef.h¿include ¡sys/syscall.h¿

include çonstants.h"include "memoryFunctions.h"

—– initializeInput —–

.text .align 2 .globl initializeInput .ent initializeInput initializeInput: .frame $fp, 16, ra.setnoreorder.cploadt9.setreorder$

Stack frame creation subu sp,sp,16

.cprestore 0 sw $fp, 12(sp)swgp, 8(sp)$

de aqui al fin de la funcion uso $fpenlugardesp.move$fp,sp

Parameters sw a0,16(fp)$Guardoenladirecciondememoria$16(fp) la variable iFileDescriptor (int). sw a1,20(fp)$Guardoenladirecciondememoria$20(fp) la variable ibytes (size$_t$).

ofd = oFileDescriptor; lw v0,16(fp)$Cargoenv0iFileDescriptor.swv0, ifdGuardoelcontenidodev0, iFileD$

obuffer.sizeBytes = obytes; lw v0,20(fp)$Cargoenv0obytes.swv0, ibuffer + 8GuardoensizeBytes(ibuffer + 8)elcontenidodev0(obytes).$

move sp,$fplw$fp,12(sp) destruyo stack frame addu sp,sp,16 vuelvo a funcion llamante j ra

.end initializeInput

Variables auxiliares

.data

—————————————— typedef struct char * buffer; int quantityCharactersInBuffer; size$_t sizeBytes; Buffer; BufferibufferBufferobuffer-$ $- -$

Variables para la parte de input

.globl ifd .section .bss TODO DESPUES VER SI ESTO SE PUEDE ELIMINAR .align 2 .type ifd, @object .size ifd, 4 ifd: .space 4

.globl lastPositionInIBufferRead .align 2 .type lastPositionInIBufferRead, @object .size lastPositionInIBufferRead, 4 lastPositionInIBufferRead: .word -1

.globl ibuffer .section .bss TODO VER SI ANDA BIEN Y SACARLO .align 2 .type ibuffer, @object .size ibuffer, 12 ibuffer: .space 12

.globl endIFile .globl endIFile .align 2 .type endIFile, @object .size endIFile, 4 endIFile: .space 4

Variables para la parte de input

.globl ofd .align 2 .type ofd, @object .size ofd, 4 ofd: .space 4

.globl obuffer .align 2 .type obuffer, @object .size obuffer, 12 obuffer: .space 12

Mensajes de error

.rdata Stack frame:

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

Figura 2: Stack frame: cleanBuffers

4.2. Código MIPS32: copyFromLexicoToOBuffer.S

```
1 CODIGO ACA
```

Stack frame:

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

Figura 3: Stack frame: copyFromLexicoToOBuffer

4.3. Código MIPS32: initializeBuffer.S

```
1 CODIGO ACA
```

Stack frame:

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

Figura 4: Stack frame: initializeBuffer

4.4. Código MIPS32: isKeywords.S

```
1 | CODIGO ACA
```

Stack frame:

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

Figura 5: Stack frame: isKeywords

4.5. Código MIPS32: loadBufferInitial.S

```
1 | CODIGO ACA
```

Stack frame:

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

Figura 6: Stack frame: loadBufferInitial

4.6. Código MIPS32: loadIBufferWithIFile.S

```
1 | CODIGO ACA
```

Stack frame:

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

Figura 7: Stack frame: loadIBufferWithIFile

4.7. Código MIPS32: loadInLexico.S

```
1 CODIGO ACA
```

Stack frame:

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

Figura 8: Stack frame: loadInLexico

4.8. Código MIPS32: myfree.S

```
1 CODIGO ACA
```

4.9. Código MIPS32: mymalloc.S

```
1 CODIGO ACA
```

4.10. Código MIPS32: myRealloc.S

```
1 CODIGO ACA
```

Stack frame:

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

Figura 9: Stack frame: myRealloc

4.11. Código MIPS32: palindrome.S

```
1 CODIGO ACA
```

Stack frame:

int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes)				
Offset	**Contents**	**Type reserved area**	**Comment**	
76	obytes			
72	ofd			
68	ibytes			
64	ifd			
60			**nothing to keep**	
56	ra	**SRA**		
52	fp			
48	gp			
44	Resultado de la función	**LTA**		
40	resultProcessWrite rdoClean		resultProcessWrite: ERROR_MEMORY | ERROR_WRITE |OKEY rdoClean: OKEY | Error	
36	rdoLoadIBuffer		OKEY_I_FILE | Se usa para guardar el resultado de la invocación a loadIBufferWithIFile.	
32	error		FALSE | TRUE	
28	rdoProcess		Puede ser igual a OKEY o resulProcessWrite	
24	*amountSavedInOBuffer			
20			**nothing to keep**	
16			**nothing to keep**	
12	a3	**ABA**	Inicialmente contiene el valor del parametro obytes. Cada vez que se invoca a SYS_write, se guarda en a3 si hubo o no error.	Invocación a loadBufferInitial: 1) isize -> a0 || ibuffer -> a1 2) osize -> a0 || obuffer -> a1 Invocación a myfree: 1) ibuffer -> a0 2) obuffr -> a0 3) *amountSavedInOBuffer -> a0 Invocación a mymalloc: 1) 4 -> a0 Invocación a loadIBufferWithIFile: 1) ibytes -> a0 || ifd -> a1 Invocación a processDataInIBuffer: 1) ibuffer -> a0 || *amountSavedInOBuffer -> a1 Invocación a initializeBuffr: 1) ibytes -> a0 || ibuffer -> a1 Invocación a cleanBuffers: 1) *amountSavedInOBuffer -> a0
8	a2		Inicialmente contiene el valor del parametro ofd.	
4	a1		Inicialmente contiene el valor del parametro ibytes.	
0	a0		Inicialmente contiene el valor del parametro ifd.	

Figura 10: Stack frame: palindrome

4.12. Código MIPS32: processDataInIBuffer.S

```
1 | CODIGO ACA
```

Stack frame:

int processDataInIBuffer(char * ibuffer, int * amountSavedInOBuffer)				
Offset	**Contents**	**Type reserved area**	**Comment**	
84	* amountSavedInOBuffer			
80	* ibuffer			
76			**nothing to keep**	
72	ra	**SRA**		
68	fp			
64	gp			
60		**LTA**	**nothing to keep**	
56	Resultado de la función		OKEY || Error	
52	rdoWrite		OKEY || Error	
48	amountToSaved			
44	itsPalindromic		FALSE || TRUE	
40	character		char character = ibuffer[idx]	
36	rdo		OKEY || LOAD_I_BUFFER	
32	idx		Inicialmente igual a 0	
28	loadIBuffer		FALSE || TRUE	
24	findEnd		FALSE || TRUE	
20			**nothing to keep**	
16			**nothing to keep**	
12	a3	**ABA**	Cada vez que se invoca a SYS_write (para informar errores) guarda en a3 si hubo o no error.	Invocación a isKeywords: 1) character -> a0 Invocación a loadInLexico: 1) character -> a0 2) '\n' -> a0 Invocación a verifyPalindromic: 1) lexico -> a0 || quantityCharactersInLexico -> a1 Invocación a myRealloc: 1) obuffer -> a0 || amountToSaved -> a1 || *amountSavedInOBuffer -> a3 Invocación a copyFromLexicoToOBuffer: 1) amountSavedInOBuffer -> a0 Invocación a writeBufferInOFile: 1) amountSavedInOBuffer -> a0 || obuffer -> a1 Invocación a myfree: 1) obuffer -> a0 2) lexico -> a0 Invocación a loadBufferInitial: 1) osize -> a0 || obuffer -> a1
8	a2			
4	a1		Inicialmente contiene el valor del parametro * amountSavedInOBuffer.	
0	a0		Inicialmente contiene el valor del parametro * ibuffer.	

Figura 11: Stack frame: processDataInIBuffer

4.13. Código MIPS32: toLowerCase.S

```
1 CODIGO ACA
```

Stack frame:

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

Figura 12: Stack frame: toLowerCase

4.14. Código MIPS32: verifyPalindromic.S

```
1 CODIGO ACA
```

Stack frame:

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

Figura 13: Stack frame: verifyPalindromic

4.15. Código MIPS32: writeBufferInOFile.S

```
1 CODIGO ACA
```

Stack frame:

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

Figura 14: Stack frame: writeBufferInOFile

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

```
$ 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).

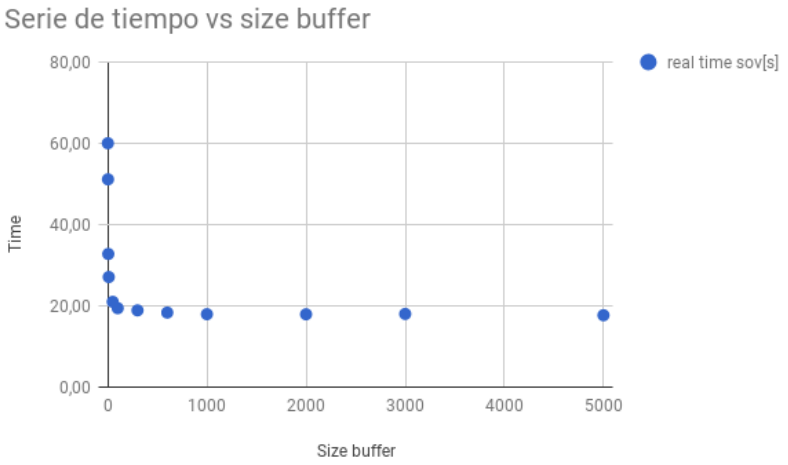

Figura 15: Gráfico de incidencia del buffer

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.

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.

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