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:

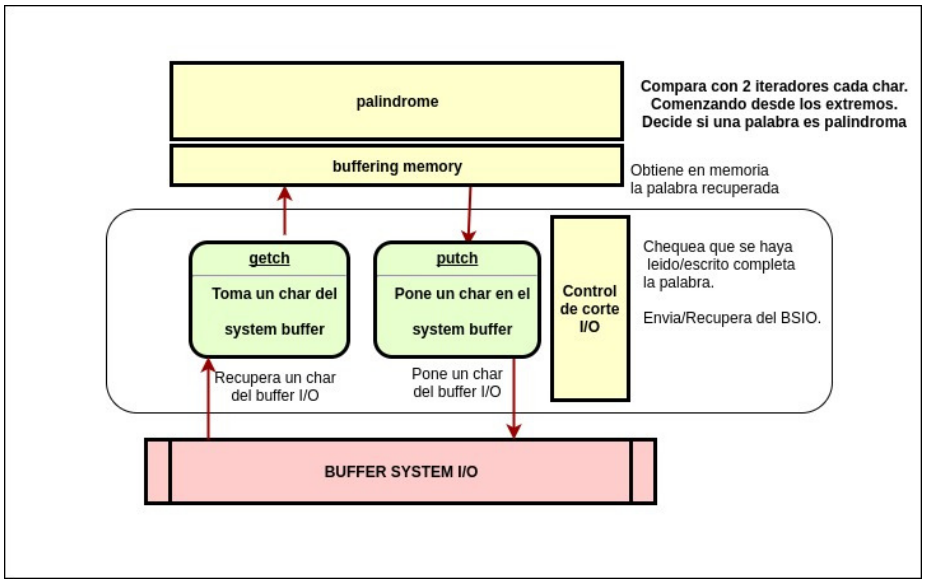

Figura 1: Diagrama: disposición palindrome

3. Implementación

3.1. Código fuente en lenguaje C: bufferFunctions.c

```
1
2
3
4
```

```
5
6   #include
7
8
9   int ifd = 0;
10  int lastPositionInIBufferRead = -1;
11  Buffer ibuffer = { NULL, 0, 0 };
12
13  int endIFile = FALSE;
14
15
16  int ofd = 0;
17  Buffer obuffer = { NULL, 0, 0 };
18
19
20  void initializeInput(int iFileDescriptor, size_t ibytes) {
21          ifd = iFileDescriptor;
22          ibuffer.sizeBytes = ibytes;
23  }
24
25  void initializeOutput(int oFileDescriptor, size_t obytes) {
26          ofd = oFileDescriptor;
27          obuffer.sizeBytes = obytes;
28  }
29
30
31
32  int loadIBufferWithIFile() {
33          if (ibuffer.buffer == NULL) {
34                  ibuffer.buffer = (char *) malloc(ibuffer.
                        sizeBytes*sizeof(char));
35                  if (ibuffer.buffer == NULL) {
36                          fprintf(stderr,
                                );
37                          return ERROR_MEMORY;
38                  }
39          }
40
41          int completeDelivery = FALSE;
42          ibuffer.quantityCharactersInBuffer = 0;
43          int bytesToRead = ibuffer.sizeBytes;
44
45
46          while (completeDelivery == FALSE && endIFile == FALSE) {
47                  int bytesRead = read(ifd, ibuffer.buffer +
                        ibuffer.quantityCharactersInBuffer,
                        bytesToRead);
48                  if (bytesRead == -1) {
49                          fprintf(stderr,
                                );
50                          return ERROR_I_READ;
51                  }
52
53                  if (bytesRead == 0) {
54                          endIFile = TRUE;
55                  }
56
57                  ibuffer.quantityCharactersInBuffer += bytesRead;
58                  bytesToRead = ibuffer.sizeBytes - ibuffer.
                        quantityCharactersInBuffer;
```

```
59
60                  if (bytesToRead <= 0) {
61                          completeDelivery = TRUE;
62                  }
63          }
64
65          lastPositionInIBufferRead = -1;
66
67          return OKEY_I_FILE;
68  }
69
70
71
72
73  int getch() {
74          if (ibuffer.buffer == NULL || lastPositionInIBufferRead
                == (ibuffer.quantityCharactersInBuffer - 1)) {
75                  if (endIFile == TRUE) {
76                          return EOF;
77                  }
78                  int resultLoadIBuffer = loadIBufferWithIFile();
79                  if (resultLoadIBuffer == ERROR_I_READ) {
80                          return ERROR_I_READ;
81                  }
82
83                  if (ibuffer.quantityCharactersInBuffer == 0) {
84                          return EOF;
85                  }
86          }
87
88          lastPositionInIBufferRead ++;
89          return ibuffer.buffer[lastPositionInIBufferRead];
90  }
91
92  int writeBufferInOFile() {
93          if (obuffer.buffer == NULL || obuffer.
                quantityCharactersInBuffer <= 0) {
94                  return OKEY;
95          }
96
97          int completeDelivery = FALSE;
98          int bytesWriteAcum = 0;
99          int bytesToWrite = obuffer.quantityCharactersInBuffer;
100         while (completeDelivery == FALSE) {
101                 int bytesWrite = write(ofd, obuffer.buffer +
                        bytesWriteAcum, bytesToWrite);
102                 if (bytesWrite < 0) {
103                         fprintf(stderr,
                                                                );
104                         return ERROR_WRITE;
105                 }
106
107                 bytesWriteAcum += bytesWrite;
108                 bytesToWrite = obuffer.
                        quantityCharactersInBuffer - bytesWriteAcum;
109
110                 if (bytesToWrite <= 0) {
111                         completeDelivery = TRUE;
112                 }
113         }
114
115         return OKEY;
```

```
116 }
117
118 int putch(int character) {
119         if (obuffer.buffer == NULL) {
120                 obuffer.buffer = (char *) malloc(obuffer.
                        sizeBytes*sizeof(char));
121                 if (obuffer.buffer == NULL) {
122                         fprintf(stderr,
                                );
123                         return ERROR_MEMORY;
124                 }
125
126                 obuffer.quantityCharactersInBuffer = 0;
127         }
128
129         obuffer.buffer[obuffer.quantityCharactersInBuffer] =
                character;
130         obuffer.quantityCharactersInBuffer ++;
131
132         if (obuffer.quantityCharactersInBuffer == obuffer.
                sizeBytes) {
133                 writeBufferInOFile();
134                 obuffer.quantityCharactersInBuffer = 0;
135         }
136
137         return OKEY;
138 }
139
140 int flush() {
141         if (obuffer.buffer != NULL && obuffer.
                quantityCharactersInBuffer > 0) {
142                 return writeBufferInOFile();
143         }
144
145         return OKEY;
146 }
147
148
149
150
151 void freeResources() {
152         if (ibuffer.buffer != NULL) {
153                 free(ibuffer.buffer);
154                 ibuffer.buffer = NULL;
155         }
156
157         if (obuffer.buffer != NULL) {
158                 free(obuffer.buffer);
159                 obuffer.buffer = NULL;
160         }
161 }
162
163
164
165
166 int loadInBuffer(char character, Buffer * buffer, size_t
        sizeInitial) {
167         if (buffer->buffer == NULL) {
168                 buffer->buffer = malloc(sizeInitial * sizeof(
                        char));
169                 buffer->sizeBytes = sizeInitial;
```

```
170         } else if (buffer->quantityCharactersInBuffer >= buffer
                ->sizeBytes) {
171             size_t bytesLexicoPreview = buffer->sizeBytes;
172
173             buffer->sizeBytes = bytesLexicoPreview * 2;
174
175             char * auxiliary = myRealloc(buffer->buffer,
                    buffer->sizeBytes*sizeof(char),
                    bytesLexicoPreview);
176             if (auxiliary == NULL) {
177                 cleanContentBuffer(buffer);
178             } else {
179                 buffer->buffer = auxiliary;
180             }
181         }
182
183         if (buffer->buffer == NULL) {
184             fprintf(stderr,
                                        );
185             return ERROR_MEMORY;
186         }
187
188         buffer->buffer[buffer->quantityCharactersInBuffer] =
                character;
189         buffer->quantityCharactersInBuffer ++;
190
191         return OKEY;
192 }
193
194 void cleanContentBuffer(Buffer * buffer) {
195         if (buffer->buffer != NULL) {
196             free(buffer->buffer);
197             buffer->buffer = NULL;
198         }
199
200         buffer->quantityCharactersInBuffer = 0;
201         buffer->sizeBytes = 0;
202 }
```

3.2. Código fuente en lenguaje C: memoryFunctions.c

```
1
2
3
4
5   #include
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
4
5
6
7
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
18  #include
19
20  #define VERSION
21
22  size_t isize = 1;
23  size_t osize = 1;
24
25  int executeHelp() {
26          fprintf(stdout,             );
27          fprintf(stdout,                     );
28          fprintf(stdout,                     );
29          fprintf(stdout,                             );
30          fprintf(stdout,               );
31          fprintf(stdout,
                                    );
```

```
32          fprintf(stdout,
                                    );
33          fprintf(stdout,
                                      );
34          fprintf(stdout,
                                       );
35          fprintf(stdout,
                                             );
36          fprintf(stdout,
                                              );
37          fprintf(stdout,                 );
38          fprintf(stdout,                                          );
39
40          return OKEY;
41  }
42
43  int executeVersion() {
44          fprintf(stdout,                       , 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
56          if (argc == 1) {
57
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
68          const char* const smallOptions =             ;
69
70
71          const struct option longOptions[] = {
72                  {           ,              no_argument,
                        0,      },
73                  {       ,                  no_argument,
                        0,      },
74                  {        ,                 required_argument,  0,
                        },
75                  {         ,                required_argument,  0,
                        },
76                  {              ,  required_argument,      0,
                        },
77                  {              ,  required_argument,      0,
                        },
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          while ((opt = getopt_long(argc, argv, smallOptions,
88                                    longOptions, &longIndex )) !=
                                         -1 && incorrectOption ==
                                         FALSE && finish == FALSE
                                        ) {
89                  switch (opt) {
90                          case     :
91                                  result = executeVersion();
92                                  finish = TRUE;
93                                  break;
94                          case     :
95                                  result = executeHelp();
96                                  finish = TRUE;
97                                  break;
98                          case     :
99                                  pathInput = optarg;
100                                 break;
101                         case     :
102                                 pathOutput = optarg;
103                                 break;
104                         case     :
105                                 iBufBytes = optarg;
106                                 break;
107                         case     :
108                                 oBufBytes = optarg;
109                                 break;
110                         default:
111                                 incorrectOption = TRUE;
112                 }
113         }
114
115         if (incorrectOption == TRUE) {
116                 fprintf(stderr,
                                );
117                 return INCORRECT_MENU;
118         }
119
120         if (finish == TRUE) {
121                 return result;
122         }
123
124         if (iBufBytes != NULL) {
125                 char *finalPtr;
126                 isize = strtoul(iBufBytes, &finalPtr, 10);
127                 if (isize == 0) {
128                         fprintf(stderr,

                                        );
129                         return ERROR_BYTES;
130                 }
131         }
132
133         if (oBufBytes != NULL) {
134                 char *finalPtr;
135                 osize = strtoul(oBufBytes, &finalPtr, 10);
136                 if (osize == 0) {
```

```
137                         fprintf(stderr,
                                                );
138                         return ERROR_BYTES;
139                 }
140         }
141
142         if (pathInput == NULL || strcmp(    ,pathInput) == 0) {
143                 inputFileDefault = TRUE;
144         }
145
146         if (pathOutput == NULL || strcmp(    ,pathOutput) == 0) {
147                 outputFileDefault = TRUE;
148         }
149
150         if (inputFileDefault == FALSE) {
151                 fileInput = fopen(pathInput,     );
152                 if (fileInput == NULL) {
153                         fprintf(stderr,
                                                , pathInput);
154                         return ERROR_FILE;
155                 }
156         }
157
158         if (outputFileDefault == FALSE) {
159                 fileOutput = fopen(pathOutput,     );
160                 if (fileOutput == NULL) {
161                         fprintf(stderr,
                                                , pathOutput);
162
163                         if (inputFileDefault == FALSE) {
164                                 int result = fclose(fileInput);
165                                 if (result == EOF) {
166                                         fprintf(stderr,
                                                , pathInput);
167                                 }
168                         }
169
170                         return ERROR_FILE;
171                 }
172         }
173
174         int ifd = fileno(fileInput);
175         int ofd = fileno(fileOutput);
176
177         int executeResult = palindrome(ifd, isize, ofd, osize);
178
179         int resultFileInputClose = 0;
180
181         if (inputFileDefault == FALSE && fileInput != NULL) {
182                 resultFileInputClose = fclose(fileInput);
183                 if (resultFileInputClose == EOF) {
184                         fprintf(stderr,
```

```
                                                        , pathInput);
185                 }
186         }
187
188         if (outputFileDefault == FALSE && fileOutput != NULL) {
189                 int result = fclose(fileOutput);
190                 if (result == EOF) {
191                         fprintf(stderr,
                                                        , pathOutput);
192                         resultFileInputClose = EOF;
193                 }
194         }
195
196         if (resultFileInputClose != 0) {
197                 return ERROR_FILE;
198         }
199
200         return executeResult;
201 }
202
203 int main(int argc, char **argv) {
204
205         if (argc > 9) {
206                 fprintf(stderr,
                                                , argc);
207                 return INCORRECT_QUANTITY_PARAMS;
208         }
209
210         return executeByMenu(argc, argv);
211 }
```

3.4. Código fuente en lenguaje C: palindromeFunctions.c

4. Código MIPS32

4.1. Código MIPS32: bufferFunctions.S

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #include
5  #include
6
7
8  ##----- initializeInput -----##
9
10         .text
11         .align          2
12         .globl          initializeInput
13         .ent            initializeInput
14 initializeInput:
15         .frame          $fp,16,ra
16         .set            noreorder
17         .cpload         t9
18         .set            reorder
19
```

```
20          # Stack frame creation
21          subu            sp,sp,16
22
23          .cprestore 0
24          sw              $fp,12(sp)
25          sw              gp,8(sp)
26
27          # de aqui al fin de la funcion uso $fp en lugar de sp.
28          move            $fp,sp
29
30          # Parameters
31          sw              a0,16($fp)              # Guardo en la
                direccion de memoria 16($fp) la variable
                iFileDescriptor (int).
32          sw              a1,20($fp)              # Guardo en la
                direccion de memoria 20($fp) la variable ibytes (
                size_t).
33
34          # ofd = oFileDescriptor;
35          lw              v0,16($fp)              # Cargo en v0
                iFileDescriptor.
36          sw              v0,ifd                  # Guardo el
                contenido de v0, iFileDescriptor, en la variable ifd
                .
37
38          # obuffer.sizeBytes = obytes;
39          lw              v0,20($fp)              # Cargo en v0
                obytes.
40          sw              v0,ibuffer+8            # Guardo en
                sizeBytes (ibuffer+8) el contenido de v0 (obytes).
41
42          move            sp,$fp
43          lw              $fp,12(sp)
44          # destruyo stack frame
45          addu            sp,sp,16
46          # vuelvo a funcion llamante
47          j   ra
48
49          .end            initializeInput
50
51
52
53
54  ## Variables auxiliares
55
56          .data
57
58
59          # ------------------------------------ #
60          #                                      #
61          # typedef struct  {                    #
62          #    char * buffer;                    #
63          #    int quantityCharactersInBuffer;   #
64          #    size_t sizeBytes;                 #
65          # } Buffer;                            #
66          #                                      #
67          # Buffer ibuffer                       #
68          # Buffer obuffer                       #
69          #                                      #
70          # ------------------------------------ #
71
72
```

```
73          ## Variables para la parte de input
74
75          .globl  ifd
76  #       .section        .bss    #TODO DESPUES VER SI ESTO SE
        PUEDE ELIMINAR
77          .align  2
78          .type   ifd, object.size ifd, 4ifd:.space 4.globl
                lastPositionInIBufferRead.align 2.type
                lastPositionInIBufferRead, object
79          .size   lastPositionInIBufferRead, 4
80  lastPositionInIBufferRead:
81          .word   -1
82
83          .globl  ibuffer
84  #       .section        .bss  TODO VER SI ANDA BIEN Y SACARLO
85          .align  2
86          .type   ibuffer, object.size ibuffer, 12ibuffer:.space 12.globl
                endIFile.globl endIFile.align 2.type endIFile, object
87          .size   endIFile, 4
88  endIFile:
89          .space  4
90
91
92          ## Variables para la parte de input
93
94          .globl  ofd
95          .align  2
96          .type   ofd, object.size ofd, 4ofd:.space 4.globl obuffer.align
                2.type obuffer, object
97          .size   obuffer, 12
98  obuffer:
99          .space  12
100
101 ## Mensajes de error
102
103         #.rdata
```

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$(fp) Cargoenv0 iFileDescriptor.swv0, ifdGuardoelcontenidodev0, iFileDe$

obuffer.sizeBytes = obytes; lw v0,20$(fp) Cargoenv0 obytes.swv0, ibuffer + 8 Guardoen sizeBytes(ibuffer + 8) elcontenidodev0(obytes)$.

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; $size_t sizeBytes; Buffer; Bufferibuffer Bufferobuffer-$

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).

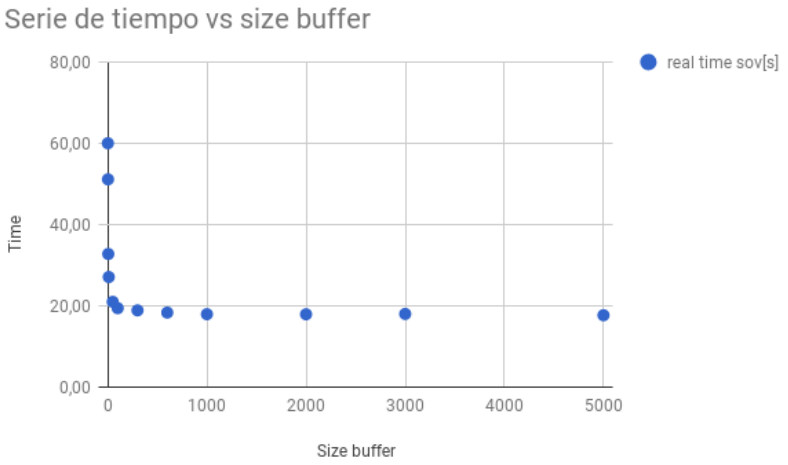

Figura 15: Gráfico de incidencia del buffer

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