Universidad de Buenos Aires

Organización del Computador (66.20)

Trabajo Práctico 1

Darius Maitia - 95436

Maximiliano Burastero - 94508

Marco Rodrigo Albanesi - 86063

Índice

1. Objetivo

El objetivo del presente trabajo práctico es elaborar una función en lenguaje Assembly (para una máquina MIPS32) respetando la ABI (application binary interface) dictada en clase. Tal función se enmarca dentro de un programa en C capaz de analizar palabras y detectar si son palíndromos o no, constituyendo la parte en assembly el núcleo central del programa.

2. Introducción

El microprocesador de una máquina MIPS32 dispone de un set de registros que si bien se constituyen de igual manera a nivel de hardware, cumplen roles distintos entre sí debido a la Application Binary Inferface. Dicha convención determina que disponemos de registros específicos para transmitir parámetros entre funciones, registros temporales, registros de retorno, etc. Además la ABI dictamina un uso específico de la memoria (particularmente del stack frame) entre funciones callers (llamadoras) y callees (llamadas).

3. Desarrollo

A modo de familiarizarnos con el uso de los registros de la CPU de una máquina con arquitectura MIPS32 y su organización del stack frame, nos proponemos desarrollar un programa principalmente en lenguaje Assembly denominado "Palindrom Finder"[1] que procesa palabras ingresadas a través de un archivo de texto o de la salida estándar y devuelve aquellas que son palíndromos o no. A modo de simplificar el programa, el alfabeto con el que trabajamos está constituido por el conjunto { a - z, A - Z, 0 - 9, - , _ }. El análisis acerca de si una palabra es palíndromo o no se realiza sin tomar en consideración las mayúsculas y minúsculas (case insensitive), no así la salida, que se escribirá tal y como fue ingresada respetando las mayúsculas.
En Assembly nos proponemos desarrollar la función palindrome, de firma:

```
int palindromes(int input_file_no, int output_file_no, size_t input_buf_size, size_t
    output_buf_size);
```

Tal función se desarrolla siguiendo el esquema siguiente:

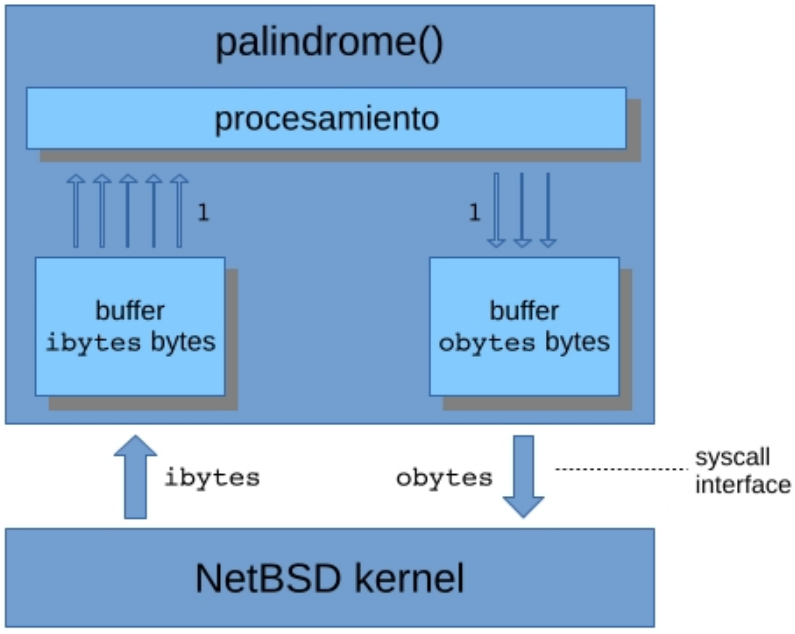

Figura 1: Diagrama de la funcion palindrome()

Los parámetros ibytes y obytes determinan de a cuántos bytes leemos de entrada (escribimos en salida), osea que disponemos de un buffer de ibytes y otro de obytes de tamaño para lectura y escritura. Adentro del módulo de procesamiento (constituido por la funcion is_palindrome) construimos un tercer buffer, extensible, con el propósito de almacenar una palabra entera de la dimensión que fuera necesaria.

[1] El código del programa se adjunta en el anexo.

La función palindrome hace llamados a varias otras funciones, todas implementadas en lenguaje Assembly y agrupadas en distintos archivos:

1. buffers.S
   a) initBuffers
   b) freeBuffers
   c) getch
   d) putch
   e) flushbuf
   f) isEof
   g) buildbuf
2. is_palindrome
3. charIsValid
4. mymalloc.S
   a) mymalloc
   b) myfree
   c) myrealloc
5. palindromes.S

En el archivo utils.h encapsulamos en forma de dos funciones de precompilador código relativo al manejo del stack entre funciones para simplicidad del código, puesto que esos segmentos de código se repiten en numerosas ocasiones.

La función initBuffers tiene la misma firma que palindromes y guarda en variables globales los números de archivo de entrada y salida, la dirección de los buffers, los tamaños, las posiciones donde se está escribiendo y leyendo y en el caso del buffer de entrada si ya se alcanzó el fin de archivo.

La función buildbuf hace invocación a la función assembly mymalloc provista en clase para reservar memoria en el heap y por otro lado la función freeBuffers invoca a la función myfree provista también en clase para liberar la memoria reservada en el heap.

getch no recibe parámetros y devuelve un caracter desde el buffer disparando una lectura del archivo cuando es necesario. putch recibe un caracter como parámetro y a escribir en el buffer de salida, cuando este se llena se escribe a un archivo (o salida estandar) mediante la función flushBuf. Ésta también es llamada por freeBuffers para escribir lo que haya quedado allí. isEof devuelve 1 si se alcanzó el fin del archivo de entrada o 0 en caso contrario.

En mymalloc.S se corrigió el caso en que la syscall mmap devuelva error mediante el registro a3, como por ejemplo cuando se pide reservar una cantidad muy grande de bytes. También se agregó la función myrealloc, que recive como parámetro el puntero y el nuevo tamaño y en caso exitoso devuelve un nuevo puntero copiando el contenido y liberando el original.

3.1. Código Assembly

3.1.1. utils.h

```
#ifndef _UTILS_H_
#define _UTILS_H_

#define SUCCESS 0
#define ERROR -1

/* Macro para construir el stack de tamano SS */
#define FRAME(SS) \
    .frame $fp, SS, ra; \
    .set noreorder; \
    .cpload t9; \
    .set reorder; \
    \
    subu sp, sp, SS; \
    .cprestore (SS-12); \
    sw $fp, (SS-8)(sp); \
    sw ra, (SS-4)(sp); \
```

```
    move $fp, sp ; \
    \
    sw a0, (SS)(sp); \
    sw a1, (SS+4)(sp); \
    sw a2, (SS+8)(sp); \
    sw a3, (SS+12)(sp)

/* Macro para destruir el stack de tamano SS y saltar a $ra */
#define RETURN(SS) \
    lw  gp, (SS-12)(sp); \
  lw  $fp, (SS-8)(sp); \
  lw  ra, (SS-4)(sp); \
  addu  sp, sp, SS; \
  jr  ra

#endif
```

3.1.2. palindromes.S

Código

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

#include "utils.h"

#define BUFFER_INITIAL_SIZE 32

# segmento de texto del programa
.text
.abicalls
.align 2

#define PALINDROMES_STACK 48

#define BUFSIZE 16
#define BUFPTR 20
#define LEN 24
#define CHAR 28
#define STATUS 32

/*
int palindromes(int input_file_no, int output_file_no, size_t input_buf_size, size_t
    output_buf_size) {
    int status = initBuffers(input_file_no, output_file_no, input_buf_size, output_buf_size)

    if (status >= 0) {
        size_t bufsize = BUFFER_INITIAL_SIZE
        char* bufptr = (char*) mymalloc(bufsize)
        if (bufptr == 0) {
            write(2, memError, memErrorLen)
            status = ERROR
        } else {
            size_t length = 0
            while(true) {
                int ch = getch()

                if (ch < 0)
                    break

                if(isvalid(ch)) {
                    if (length == bufsize) {
                        bufSize *=2
                        void* newptr = myrealloc(bufptr, bufSize)
                        if (newptr == 0) {
                            write(2, memError, memErrorLen)
                            status = ERROR
                            break
                        } else
                            bufptr = (char*) newptr
                    }

                    if (status >= 0) {
                        buf[length] = ch
                        length += 1
                    }
                } else {
```

```
                    status = checkWord(bufptr, length)
                    if (status < 0)
                        break;

                    length = 0
                }
            }

            if (! isEof)
                status = ERROR
            else
                checkWord(bufptr, length)
        }
    }

    freeBuffers()

    if (bufptr != 0)
        free(bufptr)

    return status
}
*/
.globl  palindromes
.ent  palindromes
palindromes:
    FRAME(PALINDROMES_STACK)

    sw zero, BUFPTR(sp) #bufPtr = 0

    #recibe los mismos parametros que esta funcion
    jal initBuffers
    bltz v0, retErr

    li t0, BUFFER_INITIAL_SIZE
    sw t0, BUFSIZE(sp) #bufsize

    move a0, t0
    jal mymalloc

    sw v0, BUFPTR(sp) #bufPtr

    ## if (bufPtr == 0) return ERROR
    bltz v0, pMemError

    li t0, 0
    sw t0, LEN(sp) #length

loop:
    #read char
    jal getch

    ##if (v0 < 0) break
    bltz v0, loopEnd

    sw v0, CHAR(sp) #char

    ##if(isvalid(c))
    move a0, v0
    jal charIsValid
    beqz v0, isSpace

    lw t0, BUFSIZE(sp) #bufSize
    lw t1, LEN(sp) #length

    ##if (length == bufSize)
    blt t1, t0, append

    sll t2, t0, 1 #bufSize *=2
    sw t2, BUFSIZE(sp)

    lw a0, BUFPTR(sp) #bufPtr
    move a1, t2 #newSize
    jal myrealloc

    bltz v0, pMemError
```

```
    sw v0, BUFPTR(sp) #bufptr

append:
    #buf[len] = char
    lw t1, LEN(sp) #length
    lw t2, BUFPTR(sp) #bufPtr

    addu t0, t2, t1 #next char ptr

    lw a0, CHAR(sp) #char
    sb a0, 0(t0)

    #len++
    addiu t1, t1, 1
    sw t1, LEN(sp) #length

    j loop

    ##else (notvalid)
isSpace:
    lw a0, BUFPTR(sp)    #bufptr
    lw a1, LEN(sp)    #length
    jal checkWord

    bltz v0, retErr

    #length = 0
    li t1, 0
    sw t1, LEN(sp)

    j loop

loopEnd:
    ## if (! isEof) return error
    jal isEof
    beqz v0, retErr

    #chequear si lo que quedo en el buffer es palindromo
    lw a0, BUFPTR(sp)    #bufptr
    lw a1, LEN(sp)    #length
    jal checkWord

    b cleanup

pMemError:
    #imprimir mensaje por stderr
    li  a0, 2
    la  a1, memError
  la  a2, memErrorLen
  lw  a2, 0(a2)
  li  v0, SYS_write
  syscall

retErr:
    li v0, ERROR
cleanup:
    sw v0, STATUS(sp) #status

    jal freeBuffers

    ##if (bufptr != 0) free(bufptr)
    lw a0, BUFPTR(sp) #bufPtr
    beqz a0, ret

    jal myfree
ret:
    lw v0, STATUS(sp) #status

  RETURN(PALINDROMES_STACK)
.end  palindromes


#define CHECK_WORD_STACK 32
#define INDEX 16
#define P_BUFPTR (CHECK_WORD_STACK)
#define P_LEN (CHECK_WORD_STACK+4)
```

```
/*
int checkWord(char* buf, size_t length) {
    int status
    if (len > 0) {
        if (is_palindrome(buf, length)) {
            for (int i = 0; i < length; i++) {
                char ch = buf[i]
                status = putch(ch)
                if (status < 0)
                    return status
            }
            status = putch('\n')
            if (status < 0)
                return status
        } else {
            status = 0
        }
    }
}
*/
.globl checkWord
.ent checkWord
checkWord:
    FRAME(CHECK_WORD_STACK)

    beqz a1, checkWordRet

    sw a0, P_BUFPTR(sp) #bufptr
    sw a1, P_LEN(sp) #length

    jal is_palindrome

    beqz v0, checkWordRet

    li t0, 0
    sw t0, INDEX(sp) #i

for_loop:
    lw t0, INDEX(sp) #i

    lw t1, P_BUFPTR(sp) #bufptr
    addu t1, t1, t0 #bufptr+i

    lbu a0, 0(t1)
    jal putch

    bltz v0, checkWordRet

    #++i
    lw t1, INDEX(sp)
    addiu t1, t1, 1
    sw t1, INDEX(sp)

    lw t2, P_LEN(sp) # length

    blt t1, t2, for_loop

    li a0, 10 #\n
    jal putch

checkWordRet:
    RETURN(CHECK_WORD_STACK)
.end checkWord

.rdata
.align 2
    memError:    .ascii "Error resevando memoria\n"
    memErrorLen:   .word  24
```

Stack

Palindromes

3.1.3. buffers.S

Código

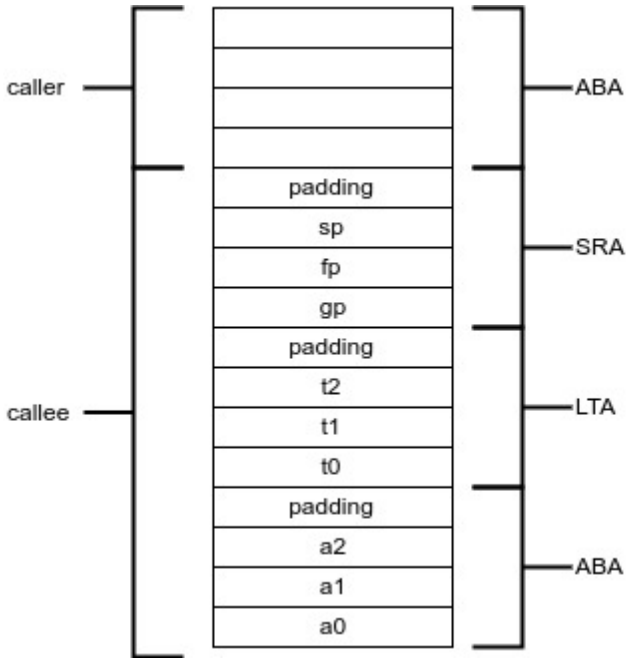

Figura 2: Stack de la funcion palindrome

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

#include "utils.h"

/*
int infile, inbuf, inbufsize, inblufcur, ineof
int outfile, outbuf, outbufsize, outbufcur;
*/
.data
.align 2
    infile:       .word 0
    inbuf:        .word 0
    inbufsize:    .word 0
    inbufcur:     .word 0
    ineof:        .word 0

    outfile:      .word 0
    outbuf:       .word 0
    outbufsize:   .word 0
    outbufcur:    .word 0

.rdata
.align 2
    mallocError:   .ascii "Error resevando memoria\n"
    readError:       .ascii "Error de lectura\n"
    writeError:      .ascii "Error de escritura\n"

#define mallocErrorLen 24
#define readErrorLen 17
#define writeErrorLen 19

.text
.abicalls
.align 2

/*
inf initBuffers(int input_file_no, int output_file_no, size_t input_buf_size, size_t
    output_buf_size) {
    infile = input_file_no
    inbufsize = input_buf_size
    inbufcur = inbufsize

    outfile = output_file_no
    outbufsize = output_buf_size
    outbufcur = 0
```

```
    int status = buildbuf(inbufsize, &inbuf)
    if (status < 0)
        return status

    status = buildbuf(outbufsize, &outbuf)

    return status
}
*/
.globl  initBuffers
.ent  initBuffers
initBuffers:
    FRAME(32)

  # Salvo los parametros como globales
  sw  a0, infile
  sw  a1, outfile

  sw  a2, inbufsize
  sw  a2, inbufcur

  sw  a3, outbufsize
  sw  zero, outbufcur

  #crear buffer lectura
  lw a0, inbufsize
  la a1, inbuf
  jal buildbuf

  bltz v0, retInitBuffers

  #crear buffer escritura
  lw a0, outbufsize
  la a1, outbuf
  jal buildbuf

retInitBuffers:
  RETURN(32)
.end    initBuffers

/*
void freeBuffers() {
    if (inbuf != 0)
        myfree(inbuf)

    if (outbuf != 0) {
        flushBuf()
        myfree(outbuf)
    }
}
*/
.globl  freeBuffers
.ent freeBuffers
freeBuffers:
    FRAME(32)

    #free inbuf
    la t0, inbuf
    lw a0, 0(t0)
    beqz a0, elif

    jal myfree
elif:
    #free outbuf
    la t0, outbuf
    lw a0, 0(t0)
    beqz a0, else

    jal flushBuf

    la t0, outbuf
    lw a0, 0(t0)
    jal myfree
else:
  RETURN(32)
.end freeBuffers
```

```
/*
int getch() {
    if (inbufcur >= inbufsize) {
        if (ineof)
            return ERROR

        inbufcur = 0

        int leido = 0
        do {
            int v0 = read(infile, inbuf+leido, inbufsize - leido)
            if (error) {
                write(2, readError, readErrorLen)
                return ERROR
            }
            leido += v0
        } while (v0 > 0 && leido < inbufsize)

        if (v0 == 0) {
            ineof = 1
            inbufsize = leido
            if (leido == 0)
                return ERROR
        }
    }

    char c = (char) *(inbuf + inbufcur++)
    return c
}
*/
.globl  getch
.ent getch
getch:
    FRAME(32)

    la t1, inbufsize
    lw t1, 0(t1)

    la t2, inbufcur
  lw t2, 0(t2)

    ##if(cur >= size)
    blt t2, t1, nextgch

    la t4, ineof
    lw t4, 0(t4)

    ##if (ateof) return ERROR
    bgtz t4, errgch

    la t2, inbufcur
    sw zero, 0(t2)

    sw zero, 16(sp) #leido

gch_loop:
    la t1, inbufsize
    lw t1, 0(t1)

    lw t2, 16(sp) #leido

    #read(infile, inbuf+leido, inbufsize-leido)
    la  a0, infile
    lw  a0, 0(a0)

    la a1, inbuf
    lw a1, 0(a1)
    addu a1, a1, t2

    move a2, t1
    subu a2, a2, t2

    li  v0, SYS_read
  syscall

    ## if(a3 == 0)
    beqz a3, chkread
```

```
    #print error
    li   a0, 2
    la   a1, readError
  li   a2, readErrorLen
  li   v0, SYS_write
  syscall

  #return ERROR
    j errgch

chkread:
    lw a2, 16(sp) #leido
    addu a2, a2, v0
    sw a2, 16(sp)

    nop
    bgtz v0, andKeepReading
    nop

    #v0 == 0 -> EOF
    li t0, 1
    la t1, ineof
    sw t0, 0(t1)

    #size = leido
    la t2, inbufsize
    sw a2, 0(t2)

    ##if(size > 0)
    bgtz a2, nextgch

    #return ERROR
    j errgch

andKeepReading: #leido < inbufsize
    la t1, inbufsize
    lw t1, 0(t1)
    blt a2, t1, gch_loop

nextgch:
    la t0, inbuf
    lw t0, 0(t0)

    la t1, inbufcur
  lw t2, 0(t1)

    # v0 = *(inbuf + inbufcur)
    addu t3, t0, t2
    lbu v0, 0(t3)

    #inbufcur++
    addiu t2, t2, 1
    la t1, inbufcur
    sw t2, 0(t1)

    j retgch

errgch:
    li v0, ERROR

retgch:
    RETURN(32)

.end getch

/*
int putch (char c) {
    if (outbufcur >= outbufsize) {
        int status = flushBuf()
        if (status < 0)
            return ERROR
    }
    *(outbuf + outbufcur++) = c
    return SUCCESS
}
*/
```

```
.globl putch
.ent putch
putch:
    FRAME(32)

    sw a0, 16(sp)

    la t1, outbufsize
    lw t1, 0(t1)

    la t2, outbufcur
    lw t2, 0(t2)

    ##if(cur >= size)
    blt t2, t1, nextpch

    jal flushBuf

    bltz v0, retpch

    la t2, outbufcur
    lw t2, 0(t2)

nextpch:
    la t0, outbuf
    lw t0, 0(t0)

    # *(outbuf + outbufcur) = a0
    lw a0, 16(sp)
    addu t3, t0, t2
    sb a0, 0(t3)

    #cur++
    addiu t2, t2, 1
    la t1, outbufcur
    sw t2, 0(t1)

    li v0, SUCCESS

retpch:
    RETURN(32)
.end putch

.globl isEof
.ent isEof
isEof:
    FRAME(8)

    la t0, ineof
    lw v0, 0(t0)

    RETURN(8)
.end isEof

/*
int flushBuf() {
    if (outbuf != 0) {
        int escrito = 0;
        int v0;
        do {
            v0 = write(outfile, outbuf+escrito, outbufcur-escrito)
            if (error) {
                write(2, writeError, writeErrorLen)
                return ERROR
            }
            escrito += v0
        } while (v0 > 0 && escrito < outbufcur)

        outbufcur = 0
        return SUCCESS
    }
    return ERROR
}
*/
.ent flushBuf
flushBuf:
    FRAME(16)
```

```
    sw zero, 0(sp) #escrito

write_do:
    lw t1, 0(sp) #escrito

    #write(outfile, outbuf+escrito, outbufcur-escrito)
    la a0, outfile
    lw a0, 0(a0)

    la a1, outbuf
    lw a1, 0(a1)
    addu a1, a1, t1

    la a2, outbufcur
    lw a2, 0(a2)
    subu a2, a2, t1

    li v0, SYS_write
    syscall

    ## if(a3 == 0)
    beqz a3, chkwrite

    #print error
    li  a0, 2
    la  a1, writeError
    li  a2, writeErrorLen
    li  v0, SYS_write
    syscall

    #return ERROR
    li v0, ERROR
    j retFB

chkwrite:
    lw t0, 0(sp) #escrito += v0
    addu t0, t0, v0
    sw t0, 0(sp)

    #v0 > 0
    bgtz v0, andKeepWriting

    #return ERROR
    li v0, ERROR
    j retFB

andKeepWriting:
    la t1, outbufcur
    lw t1, 0(t1)

    # escrito < outbufcur
    blt t0, t1, write_do

    #outbufcur = 0
    la t1, outbufcur
    sw zero, 0(t1)

    li v0, SUCCESS
retFB:
    RETURN(16)

.end flushBuf


/*
int buildbuf(size_t bufsize, void** buf) {
    void* v0 = mymalloc(bufsize)
    if (v0 == 0) {
        write(2, mallocError, mallocErrorLen);
        *buf = 0
        return ERROR
    }

    *buf = v0
    return SUCCESS
}
*/
```

```
.ent buildbuf
buildbuf:
    FRAME(40)
    sw a0, 16(sp) #tamanio
    sw a1, 20(sp) #direccion

    jal mymalloc

    lw a1, 20(sp)
    beqz v0, mallocErr

    sw v0, 0(a1)
    li v0, SUCCESS
    j retBB

mallocErr:
    #Imprimo mensaje de error
    li  a0, 2
    la  a1, mallocError
  li  a2, mallocErrorLen
  li  v0, SYS_write

  syscall

  lw a1, 20(sp)
    sw zero, 0(a1)
    li v0, ERROR

retBB:
    RETURN(40)

.end buildbuf
```

Stack

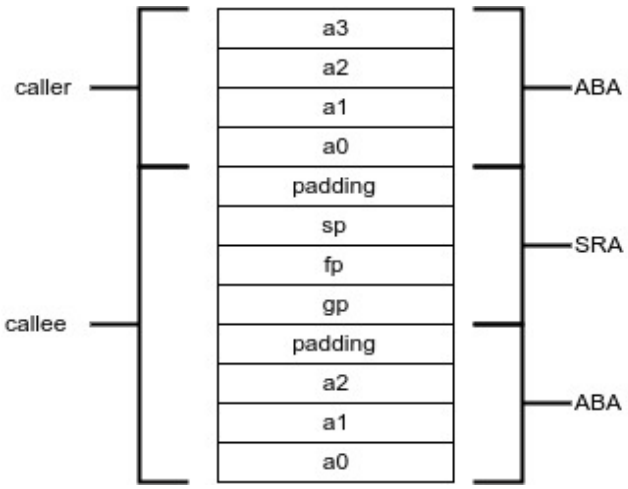

Figura 3: Stack de la funcion initBuffers

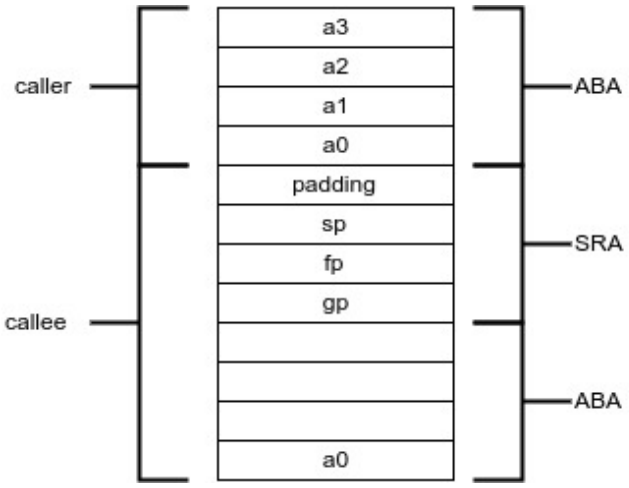

Figura 4: Stack de la funcion freeBuffers

3.1.4. charIsValid.S

Código

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

    ##   |----------------|   Se trata de una funcion hoja, no invoca a ninguna otra
    ##   |     caller     |   Por eso el stack frame mide 8
    ## 8 |----------------|   >>>
    ##   |       gp       |
    ## 4 |----------------|       Saved Registers Area
    ##   |       fp       |
    ## 0 |----------------|   <<<


    ##//charIsValid
    ##//Devuelve true si el caracter pertenece a los conjuntos
    ##// {a-z} o {A-Z} o {-} o {_}, false en caso contrario.
    ##bool charIsValid(int c) {
    ##  return ((c <= 122 && c >= 97) || (c <= 90 && c >= 65)
    ##      || (c <= 57 && c >= 48) || (c == 45) || (c == 95));
    ##}

  .text
    .abicalls
  .align 2
  .globl  charIsValid
  .ent  charIsValid

charIsValid:
    .frame $fp, 8, ra

    .set noreorder
    .cpload t9
    .set reorder
    subu sp, sp, 8
  .cprestore 4
    sw $fp,0(sp)

    move $fp,sp             ## trabajo con el frame pointer ahora

    ## guardo los argumentos en la ABA de la funcion llamante
    sw a0,8($fp)            ## int c

IF1:    li t1,122
        ble a0,t1,IF2
        b IF3

IF2:    li t1,97
        bge a0,t1,returnTrue

IF3:    li t1,90
```

```
        ble a0,t1,IF4
        b IF5

IF4:    li t1,65
        bge a0,t1,returnTrue

IF5:    li t1,57
        ble a0,t1,IF6
        b IF7

IF6:    li t1,48
        bge a0,t1,returnTrue

IF7:    li t1,45
        beq a0,t1,returnTrue

IF8:    li t1,95
        beq a0,t1,returnTrue
        b returnFalse

returnTrue:
    li v0,1
    b return

returnFalse:
    li v0,0
    b return

return:
    lw gp,4(sp)
    lw $fp,0(sp)
    addi sp,sp,8
    jr ra

.end charIsValid
```

Stack

Se trata de una función hoja que no requiere espacio para LTA ni menos ABA.

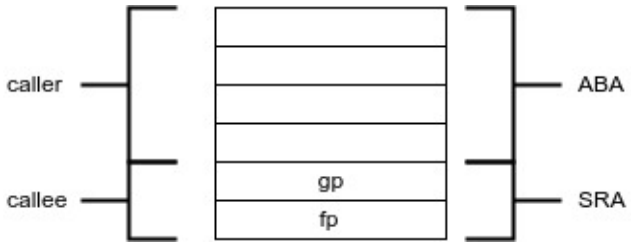

Figura 5: Stack de la funcion charIsValid

3.1.5. is_palindrome.S

Código

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

##   |----------------|   Se trata de una funcion hoja, no invoca a ninguna otra
##   |     caller     |   Por eso el stack frame mide 8
## 8 |----------------|   >>>
##   |       gp       |
## 4 |----------------|       Saved Registers Area
##   |       fp       |
## 0 |----------------|   <<<


##Codigo C
##bool is_palindrome(char *buff, size_t len) {
##    if (len == 0)
##        return false;
##
##    for (int i = 0; i < len / 2; ++i) {
```

```
##      if (tolower(buff[i]) != tolower(buff[len - 1 - i]))
##          return false;
##    }
##    return true;
##}
##


  .text ## segmento de texto del programa
        ## lo que sigue a continuacion son instrucciones
             ## y tiene que ser parte del code segment

    .abicalls       ## le decimos al compilador que vamos a usar llamadas
                    ## respetando la ABI

  .align 2    ## alineacion 2^2

  .globl  is_palindrome   ## damos a la funcion palindrome un scope global

  .ent  is_palindrome   ## el label palindrome es la direccion de la primera
                        ## instruccion de la funcion


is_palindrome:
    .frame $fp, 8, ra

    .set noreorder
    .cpload t9
    .set reorder

    subu sp, sp, 8

    .cprestore 4     ## inserta aqui "sw gp, 4(sp)",
               ## mas "lw gp, 4(sp)" luego de cada jal.

    sw $fp,0(sp)

    move $fp,sp             ## trabajo con el frame pointer ahora

    ## guardo los argumentos en la ABA de la funcion llamante
    sw a0,8($fp)            ## char * buff
    sw a1,12($fp)           ## size_t len

    beqz a1,return_false    ## if len == 0

    li t0,0                  ## i <=> t0, i = 0
    li t1,2
    div t2,a1,t1             ## t2 = len/2

L1: addiu t3,a1,-1           ## (len - 1)
    subu t3,t3,t0            ## (len - 1) - i
    addu t4,a0,t0            ## t4 = puntero a char x izquierda; t4 = buff + i
    addu t5,a0,t3            ## t5 = puntero a char x derecha; t5 = buff + [ len - 1 - i ]
    lbu t6,0(t4)              ## t6 <=> buff[i]
    lbu t7,0(t5)              ## t7 <=> buff[len - 1 - i]

##Indistinguimos mayusculas y minusculas
##Dado que los caracteres vienen validados sabemos que un caracter mayor a 97 en la tabla ascii
     es una letra mayuscula, con lo que tal comparacion es suficiente y no necesitamos acotar
    superiormente.

    addiu t8,zero,97      ##t8 = a(97)
IF1:bge t6,t8,uppercase1  ##si t6 >= a en ascii
    b IF2
uppercase1:                     ##Las minusculas pasan a ser mayusculas
    addiu t6,t6,-32             ##32 es la diferencia entre mayusculas y minusculas

IF2:bge t7,t8,uppercase2        ##Analogo pero con t7
    b compare
uppercase2:
    addiu t7,t7,-32

compare:
    bne t6,t7,return_false
    addiu t0,t0,1
    bge t0,t2,return_true
    b L1
```

```
return_false:
    li v0, 0
    b return

return_true:
    li v0, 1
    b return

return:
    lw gp,4(sp)
    lw $fp,0(sp)
    addi sp,sp,8
    jr ra

    .end is_palindrome
```

Stack

Se trata de una función callee hoja que no requiere espacio para LTA ni menos ABA (la callee).

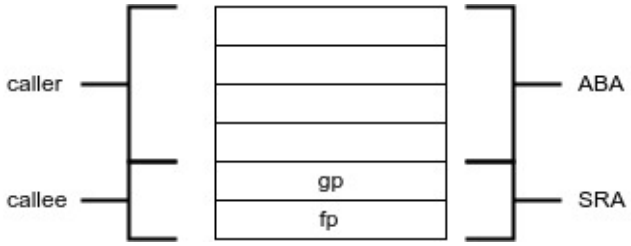

Figura 6: Stack de la funcion is_palindrome

3.1.6. mymalloc.S

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

#define MYMALLOC_SIGNATURE 0xdeadbeef

#ifndef PROT_READ
#define PROT_READ 0x01
#endif

#ifndef PROT_WRITE
#define PROT_WRITE 0x02
#endif

#ifndef MAP_PRIVATE
#define MAP_PRIVATE 0x02
#endif

#ifndef MAP_ANON
#define MAP_ANON 0x1000
#endif

  .text
  .align  2
  .globl  mymalloc
  .ent  mymalloc
mymalloc:
  subu  sp, sp, 56
  sw  ra, 48(sp)
  sw  $fp, 44(sp)
  sw  a0, 40(sp)  # Temporary: original allocation size.
  sw  a0, 36(sp)  # Temporary: actual allocation size.
  li  t0, -1
  sw  t0, 32(sp)  # Temporary: return value (defaults to -1).
#if 0
  sw  a0, 28(sp)  # Argument building area (#8?).
  sw  a0, 24(sp)  # Argument building area (#7?).
  sw  a0, 20(sp)  # Argument building area (#6).
  sw  a0, 16(sp)  # Argument building area (#5).
  sw  a0, 12(sp)  # Argument building area (#4, a3).
```

```
  sw   a0,  8(sp)  # Argument building area (#3, a2).
  sw   a0,  4(sp)  # Argument building area (#2, a1).
  sw   a0,  0(sp)  # Argument building area (#1, a0).
#endif
  move  $fp, sp

  # Adjust the original allocation size to a 4-byte boundary.
  #
  lw   t0, 40(sp)
  addiu t0, t0, 3
  and t0, t0, 0xfffffffc
  sw   t0, 40(sp)

  # Increment the allocation size by 12 units, in order to
  # make room for the allocation signature, block size and
  # trailer information.
  #
  lw   t0, 40(sp)
  addiu t0, t0, 12
  sw   t0, 36(sp)

  # mmap(0, sz, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANON, -1, 0)
  #
  li   v0, SYS_mmap
  li   a0, 0
  lw   a1, 36(sp)
  li   a2, PROT_READ|PROT_WRITE
  li   a3, MAP_PRIVATE|MAP_ANON

  # According to mmap(2), the file descriptor
  # must be specified as -1 when using MAP_ANON.
  #
  li   t0, -1
  sw   t0, 16(sp)

  # Use a trivial offset.
  #
  li   t0, 0
  sw   t0, 20(sp)

  # XXX TODO.
  #
  sw   zero, 24(sp)
  sw   zero, 28(sp)

  # Excecute the syscall, save the return value.
  #
  syscall
  sw   v0, 32(sp)
  beqz  v0, mymalloc_failed
  bnez    a3, mymalloc_failed

  # Success. Check out the allocated pointer.
  #
  lw   t0, 32(sp)
  li   t1, MYMALLOC_SIGNATURE
  sw   t1, 0(t0)

  # The actual allocation size goes right after the signature.
  #
  lw   t0, 32(sp)
  lw   t1, 36(sp)
  sw   t1,  4(t0)

  # Trailer information.
  #
  lw   t0, 36(sp) # t0: actual allocation size.
  lw   t1, 32(sp) # t1: Pointer.
  addu  t1, t1, t0 # t1 now points to the trailing 4-byte area.
  xor t2, t0, MYMALLOC_SIGNATURE
  sw   t2, -4(t1)

  # Increment the result pointer.
  #
  lw   t0, 32(sp)
  addiu t0, t0, 8
  sw   t0, 32(sp)
```

```
  b mymalloc_return

mymalloc_failed:
    sw zero, 32(sp)

mymalloc_return:
  # Restore the return value.
  #
  lw  v0, 32(sp)

  # Destroy the stack frame.
  #
  move  sp, $fp
  lw  ra, 48(sp)
  lw  $fp, 44(sp)
  addu  sp, sp, 56

  j ra
  .end  mymalloc

  .globl  myfree
  .ent  myfree
myfree:
  subu  sp, sp, 40
  sw  ra, 32(sp)
  sw  $fp, 28(sp)
  sw  a0, 24(sp)  # Temporary: argument pointer.
  sw  a0, 20(sp)  # Temporary: actual mmap(2) pointer.
  move  $fp, sp

  # Calculate the actual mmap(2) pointer.
  #
  lw  t0, 24(sp)
  subu  t0, t0, 8
  sw  t0, 20(sp)

  # XXX Sanity check: the argument pointer must be checked
  # in before we try to release the memory block.
  #
  # First, check the allocation signature.
  #
  lw  t0, 20(sp) # t0: actual mmap(2) pointer.
  lw  t1, 0(t0)
  bne t1, MYMALLOC_SIGNATURE, myfree_die

  # Second, check the memory block trailer.
  #
  lw  t0, 20(sp) # t0: actual mmap(2) pointer.
  lw  t1, 4(t0)  # t1: actual mmap(2) block size.
  addu  t2, t0, t1 # t2: trailer pointer.
  lw  t3, -4(t2)
  xor t3, t3, t1
  bne t3, MYMALLOC_SIGNATURE, myfree_die

  # All checks passed. Try to free this memory area.
  #
  li  v0, SYS_munmap
  lw  a0, 20(sp) # a0: actual mmap(2) pointer.
  lw  a1, 4(a0)  # a1: actual allocation size.
  syscall

  # Bail out if we cannot unmap this memory block.
  #
  bnez  v0, myfree_die

  # Success.
  #
  j myfree_return

myfree_die:
  # Generate a segmentation fault by writing to the first
  # byte of the address space (a.k.a. the NULL pointer).
  #
  sw t0, 0(zero)

myfree_return:
```

```
  # Destroy the stack frame.
  #
  move  sp, $fp
  lw  ra, 32(sp)
  lw  $fp, 28(sp)
  addu  sp, sp, 40

  j ra
  .end  myfree

.globl  myrealloc
.ent  myrealloc
myrealloc:

  subu  sp, sp, 48
  sw  ra, 40(sp)
  sw  $fp, 36(sp)
  move  $fp, sp

    sw  a0, 32(sp)  # argument pointer.

    lw t0, -4(a0)
    sw  t0, 28(sp)  # argument old_size.

    sw  a1, 24(sp)  # argument new_size.

    move a0, a1
    la t9, mymalloc
    jalr t9

    ## if (new ptr == 0) return 0
    beqz v0, myrealloc_return

    sw v0, 20(sp)   # new ptr

    #copia
    lw t0, 32(sp)    #old ptr
    move t1, v0     # new ptr

    lw t2, 28(sp)    #old size
    addu t2, t0, t2  #end old buf

myrealloc_do:
    lb t3, 0(t0) # char aux
    sb t3, 0(t1)

    addiu t0, 1
    addiu t1, 1

    #while t0 < t2
    bne t0, t2, myrealloc_do

    #free old ptr
    lw a0, 32(sp)

    la t9, myfree
    jalr t9

    lw v0, 20(sp)   # new ptr
    j myrealloc_return

myrealloc_return:
  # Destroy the stack frame.
  move  sp, $fp
  lw  ra, 40(sp)
  lw  $fp, 36(sp)
  addu  sp, sp, 48
  j ra
.end  myrealloc
```

4. Instrucciones de uso

4.1. Instalación del programa

Para instalar el programa sencillamente tenemos que compilar el programa de la siguiente manera:

```
./build.sh
```

4.2. Modo de uso del programa

Si ejecutamos el programa con la opción -h (o –help) nos aparecen las instrucciones de uso:

```
Usage:
        tp1 -h
        tp1 -V
        tp1 [options]
Options:
        -V, --version       Print version and quit.
        -h, --help          Print this information.
        -i, --input         Location of the input file.
        -o, --output        Location of the output file.
        -I, --ibuf-bytes    Byte-count of the input buffer.
        -O, --obuf-bytes    Byte-count of the output buffer.
Examples:
        tp1 -i ~/input -o ~/output
        cat input | tp1 -i - -o -
```

También se puede trabajar directamente con las entradas y salidas estándar. Para eso sencillamente podemos no introducir las rutas de archivos de entrada y/o salida o también podemos colocar un guion en su lugar.

5. Pruebas

Para garantizar el buen funcionamiento del programa, ejecutamos una serie de pruebas.

Creamos un script SH en el cual simulamos distintos casos de uso del programa. Por un lado tenemos el archivo de entrada que usa el usuario, y por el otro el archivo de salida que esperamos que nos devuelva el programa. Lo que hacemos es correr el programa con los mencionados archivos de entrada y guardamos la salida de estas pruebas para compararlas mediante el comando diff [3] con los archivos de salida esperados para cada caso. Si las pruebas resultaron exitosas, todos los archivos deberían ser iguales.

A continuación se muestra el contenido del archivo **tests.sh**:

```
for file in tests/input*.txt
do
  name=${file##*/input}
  name=${name%.txt}
  ./tp1 -i $file -o "tests/output${name}Test.txt" -I32 -O32
  diff -s "tests/output${name}Test.txt" "tests/output${name}Expected.txt"
done
```

A continuación se listan las pruebas realizadas:

5.1. Prueba 1

- Entrada

  Somos los primeros en completar el TP 0.

  Ojo que La fecha de entrega del TP0 es el martes 12 de septiembre.

- Salida

  ```
  Somos
  0
  Ojo
  ```

5.2. Prueba 2

- Entrada

  Es una manada de animales del fútbol vestidos de rojo y blanco, con la corona de ser los campeones de la última liga francesa, los reyes de dicha selva, aunque no tenga un león. Porque les alcanza con un Tigre y sus laderos. De la mano de una Radamel Falcao intratable, autor de dos goles, el Monaco goleó por 6-1 al Olympique Marsella y sigue firme, con puntaje perfecto, igual que el PSG, el equipo de los millones...

  Glik, a los dos minutos, puso arriba al dueño de casa, y Falcao, primero de penal y al toque de cabeza, estiro el tablero a tres goles de diferencia cuando iba poco más de media hora de juego. Ahí se terminó. Para colmo, Diakhaby y Sidibé también hicieron los suyos para el 5-0. El Olympique apenas reaccionó con el descuento de Cabella, pero Fabinho, de penal, volvió a poner los cinco de distancia.

- Salida

f
y
n
y
sus
y
a
o
y
y
a
m
s
y
n
a

5.3. Prueba 3

- Entrada

  Finalmente, el ex presidente, Carlos Saúl Menem, fue habilitado para ser candidato en las elecciones generales de octubre y poder renovar, en caso de que gane, la banca que ocupa en el senado de la Nación en representación de la provincia de La Rioja.

  La habilitación del ex mandatario, que había sido cuestionada y puesta en duda hasta el día de hoy, llega luego de que la Cámara Nacional Electoral (CNE) saliera a aclarar que "la postulación del riojano quedó actualmente habilitada".

  Esta decisión se da tras conocer el fallo de la Corte Suprema, que declaró nula la decisión del tribunal electoral de impedir a Menem presentarse en estas elecciones.

- Salida

l
Menem
y
n
n
n
a
y
d
a
C
a
n
n
n
a
Menem

5.4. Prueba 4

- Entrada

  Palabras sin sentido: hosoh fdfdjk ldldldl g-g hola_aloh u_a_u ll %s no si9is

- Salida

```
hosoh
ldldldl
g-g
hola_aloh
u_a_u
ll
s
si9is
```

5.5. Prueba 5

- Entrada

  EXE arenera, arepera, anilina, oso, ananá fruta cualquier cosa

- Salida

```
EXE
arenera
arepera
anilina
oso
```

5.6. Prueba 6

- Entrada

  neuquen palabralargaquenoentraenlos32bytesquetieneelbufferinicialmenteporloquedebeexpandirsedosveceshastaalcanzar
  reconocer

- Salida

```
neuquen
reconocer
```

5.7. Prueba 7

- Entrada El archivo tests/input7.txt No se incluye por ser de aproximadamente 600KB. Contiene palabras de diccionario en español omitiendo las que tienen caracteres que son considerados espacios por este programa.
- Salida El archivo tests/output7Expected.txt
- Performance: Para esta prueba se corrió varias veces el programa usando distintos tamaños de buffer de entrada y salida. Además de probar el correcto funcionamiento del programa, sirve para comprobar como el tamaño de los buffer repercute en la cantidad de syscalls. Mediante el programa time se ve como cambia el tiempo de ejecución y la proporción de este que es de usuario o de sistema:

  A continuación se muestra el contenido del archivo **bench.sh**:

```
bufsizes=( 1 1 1 10 10 1 100 100 1000 1000 )

for ((i=0; i<${#bufsizes[@]}; i+=2)); do
  echo "input buffer ${bufsizes[i]} bytes - output buffer ${bufsizes[i+1]} bytes:"
  time ./tp1 -I ${bufsizes[i]} -O ${bufsizes[i+1]} -i tests/input7.txt -o out
  diff tests/output7Expected.txt out
  echo ""
done
```

  Este es el resultado:

```
input buffer 1 byte - output buffer 1 bytes:

real    0m10.367s
user    0m1.215s
sys     0m9.152s
input buffer 1 byte - output buffer 10 bytes:

real    0m10.383s
user    0m1.168s
sys     0m9.211s
input buffer 10 byte - output buffer 1 bytes:

real    0m1.598s
user    0m0.730s
sys     0m0.867s
input buffer 100 byte - output buffer 100 bytes:

real    0m0.617s
user    0m0.543s
sys     0m0.074s
input buffer 1000 byte - output buffer 1000 bytes:

real    0m0.535s
user    0m0.527s
sys     0m0.008s
root@:~/shared# ./bench.sh
input buffer 1 bytes - output buffer 1 bytes:

real    0m10.102s
```

```
user    0m1.043s
sys     0m9.050s

input buffer 1 bytes - output buffer 10 bytes:

real    0m10.250s
user    0m1.035s
sys     0m9.211s

input buffer 10 bytes - output buffer 1 bytes:

real    0m1.598s
user    0m0.695s
sys     0m0.891s

input buffer 100 bytes - output buffer 100 bytes:

real    0m0.613s
user    0m0.539s
sys     0m0.074s

input buffer 1000 bytes - output buffer 1000 bytes:

real    0m0.566s
user    0m0.558s
sys     0m0.024s
```

6. Conclusiones

Escribir código assembly para la arquitectura MIPS32 nos induce a entender cuál es el uso apropiado de cada uno de los registros de la CPU y cómo manejar adecuadamente el stack entre llamadas a funciones. La convención que seguimos (la ABI de MIPS32) nos garantiza una buena comunicación entre las distintas funciones escritas en distintas unidades de compilación, logrando que tales funciones puedan interactuar entre ellas pero sin interferir unas con otras.

Además con el desarrollo de la función palindrome pudimos hacer un análisis de eficiencia relacionado con las SYSCALLS; al implementar entrada y salida mediante buffers de tamaño parametrizable se pudo ver el costo que conlleva las llamadas a sistema y la diferencia en tiempos de ejecución entre leer y escribir byte por byte o de tiras de bytes, resultando muy ineficiente la lectura byte a byte.

7. Anexo

7.1. Codigo C

7.1.1. main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <getopt.h>
#include <ctype.h>

#define ERROR -1
#define SUCCESS 0

//procesa el archivo input_file palabra por palabra y escribe los resultados en output_file.
extern int palindromes(int input_file_no, int output_file_no, size_t input_buf_size, size_t
    output_buf_size);

void show_help_menu() {
  printf("%s", "Usage:\n\t"
      "tp1 -h\n\t"
      "tp1 -V\n\t"
      "tp1 [options]\n"
      "Options:\n\t"
      "-V, --version \tPrint version and quit.\n\t"
      "-h, --help \tPrint this information.\n\t"
      "-i, --input \tLocation of the input file.\n\t"
      "-o, --output \tLocation of the output file.\n\t"
            "-I, --ibuf-bytes\t Byte-count of the input buffer.\n\t"
            "-O, --obuf-bytes\t Byte-count of the output buffer.\n\t"
      "Examples:\n\t"
      "tp1 -i ~/input -o ~/output\n");
}

int openFile(FILE** file, const char *path, const char *mode) {
  *file = fopen(path, mode);
  if (*file == NULL) {
    perror("Error opening file");
    return ERROR;
  }
  return SUCCESS;
}

int closeFile(FILE* file) {
  if (file == NULL)
    return ERROR;

  if (fclose(file) == ERROR) {
    perror("ERROR closing file");
    return ERROR;
  }
  return SUCCESS;
}

int main(int argc, char *argv[]) {
  char* input_path = 0;
  char* output_path = 0;
  size_t input_buf_size = 1;
  size_t output_buf_size = 1;

  static struct option longOpts[] = {
      {"version", 0, NULL, 'V'},
      {"help", 0, NULL, 'h'},
      {"input", 1, NULL, 'i'},
      {"output", 1, NULL, 'o'},
        {"ibuf-bytes", 1, NULL, 'I'},
      {"obuf-bytes", 1, NULL, 'O'},
      {NULL, 0, NULL, 0}
    };

    int c;
    int index = 0;
    do {
      c = getopt_long(argc, argv, "hVi:o:I:O:", longOpts, &index);
```

```
      switch (c) {
      case 'V':
          printf("%s","Palindrom Finder: v1.0\n");
          return SUCCESS;
      case 'h':
          show_help_menu();
          return SUCCESS;
      case 'i':
        input_path = optarg;
        break;
      case 'o':
        output_path = optarg;
        break;
      case 'I':
        input_buf_size = strtoul(optarg, 0, 10);
            if (input_buf_size == 0) {
                fprintf(stderr, "Valor invalido para el parametro ibuf-bytes\n");
                return ERROR;
            }
        break;
      case 'O':
        output_buf_size = strtoul(optarg, 0, 10);
            if (output_buf_size == 0) {
                fprintf(stderr, "Valor invalido para el parametro obuf-bytes\n");
                return ERROR;
            }
        break;
      case '?':
          show_help_menu();
          return ERROR;
      default:
        break;
      }
    } while (c != -1);

    int status = SUCCESS;

    FILE* input_file;
    FILE* output_file;

  if (status == SUCCESS && input_path && strcmp(input_path, "-") != 0)
    status = openFile(&input_file, input_path, "r");
  else
    input_file = stdin;

  if (status == SUCCESS && output_path && strcmp(output_path, "-") != 0)
    status = openFile(&output_file, output_path, "w");
  else
    output_file = stdout;

  if (status != ERROR)
    status = palindromes(fileno(input_file), fileno(output_file), input_buf_size,
        output_buf_size);

  closeFile(input_file);
  closeFile(output_file);

  return status;
}
```

Referencias

[1] GXemul http://gxemul.sourceforge.net/

[2] NetBDS https://www.netbsd.org/

[3] diff https://www.gnu.org/software/diffutils/diffutils.html

8. Enunciado

66:20 Organización de Computadoras
Trabajo práctico #1: programación MIPS
2^o cuatrimestre de 2017

`$Date: 2017/09/19 09:29:47 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en la sección 4.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, por escrito, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 5), la presentación de los resultados obtenidos explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

4. Descripción

En este trabajo, se reimplementará parcialmente en assembly MIPS el programa desarrollado en el trabajo práctico anterior.

Para esto, se requiere reescribir el programa, de forma tal que quede organizado de la siguiente forma:

- Arranque y configuración del programa: procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores. Desde aquí se invocará a las función de procesamiento del *stream* de entrada.

- Procesamiento: contendrá el código MIPS32 assembly con la función `palindrome()`, encargada de identificar, procesar e imprimir los componentes léxicos que resulten ser palíndromos, de forma equivalente a lo realizado en el TP anterior.

La función MIPS32 `palindrome()` antes mencionada se corresponderá con el siguiente prototipo en C:

```
int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes);
```

Esta función es invocada por el módulo de arranque y configuración del programa, y recibe en `ifd` y `ofd` los descriptores abiertos de los archivos de entrada y salida respectivamente.

Los parámetros `ibytes` y `obytes` describen los tamaños en bytes de las unidades de transferencia de datos desde y hacia el kernel de NetBSD, y permiten implementar un esquema de *buffering* de estas operaciones de acuerdo al siguiente diagrama:

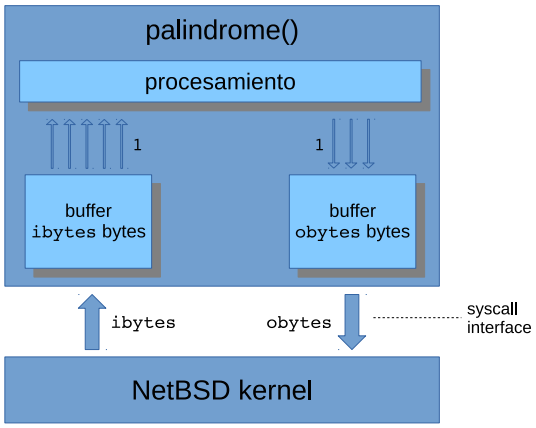

Figura 1: arquitectura de procesamiento.

Como puede verse en la figura 1, la lógica de procesamiento de la función `palindrome()` va leyendo los caracteres del *buffer* de entrada en forma individual.

En el momento en el cual `palindrome()` intente extraer un nuevo caracter, y el *buffer* de entrada se encuentre vacío, deberá ejecutar una llamada al sistema operativo para realizar una lectura en bloque y llenar completamente el buffer, siendo el tamaño de bloque igual a `ibytes` bytes.

De forma análoga, `palindrome()` irá colocando uno a uno los caracteres de las palabras capicúa en el *buffer* de salida. En el momento en el que se agote su capacidad, deberá vaciarlo mediante una operación de escritura hacia el kernel de NetBSD para continuar luego con su procesamiento.

Al finalizar la lectura y procesamiento de los datos de entrada, es probable que exista información esperando a ser enviada al sistema operativo. En ese caso `palindrome()` deberá ejecutar una última llamada al sistema con el fin de vaciar completamente el *buffer* de salida.

Se sugiere encapsular la lógica de *buffering* de entrada/salida con funciones, `getch()` y `putch()`. Asimismo durante la clase del martes 19/9 explicaremos la función `mymalloc()` que deberá ser usada para reservar dinámicamente la memoria de los buffers.

4.1. Ejemplos

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 [options]
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information.
  -i, --input       Location of the input file.
  -o, --output      Location of the output file.
  -I, --ibuf-bytes  Byte-count of the input buffer.
  -O, --obuf-bytes  Byte-count of the output buffer.
Examples:
  tp0 -i ~/input -o ~/output
```

Codificamos un archivo vacío (cantidad de bytes nula):

```
$ touch /tmp/zero.txt
$ tp0 -i /tmp/zero.txt -o /tmp/out.txt
$ ls -l /tmp/out.txt
-rw-r--r--  1 user group 0 2017-03-19 15:14 /tmp/out.txt
```

Leemos un *stream* cuyo único contenido es el caracter ASCII `M`,

```
$ echo Hola M | tp0
M
```

Observar que la salida del programa contiene aquellas palabras de la entrada que sean palíndromos (`M` en este caso).

Veamos que sucede al procesar archivo de mayor complejidad:

```
$ cat entrada.txt
Somos los primeros en completar el TP 0.

Ojo que La fecha de entrega del TP0 es el martes 12 de septiembre.
$ tp0 -i entrada.txt -o -
Somos
Ojo
```

4.2. Interfaz

A fin de facilitar la corrección y prueba de los TPs, normalizaremos algunas de las opciones que deberán ser provistas por el programa:

- `-i`, o `--input`, permite especificar la ubicación del archivo de entrada, siendo `stdin` o cuando el argumento es "`-`", o bien cuando no haya sido especificado explícitamente en la línea de comandos (valor por defecto).
- `-o`, o `--output`, para definir la ubicación del archivo de salida en forma análoga al punto anterior. Por defecto, el programa deberá escribir sobre `stdout`. Lo mismo sucederá cuando el argumento pasado es "`-`".
- `-I`, o `--ibuf-bytes` determina el tamaño en bytes del *buffer* de entrada. El valor por defecto a usar es `1`.
- `-O`, o `--obuf-bytes` nos permite dimensionar el *buffer* se salida. El valor por defecto también es `1`.

5. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño e implementación del programa.
- Comandos para compilar el programa.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, el cual también deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas).
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

6. Fechas

- Entrega: 26/9/2017.
- Vencimiento: 10/10/2017.