Informe TP1

Cotarelo Rodrigo, *Padrón Nro. 98577*
cotarelorodrigo@gmail.com

Etchegaray Rodrigo, *Padrón Nro. 96856*
rorroeche@gmail.com

Longo Nicolás, *Padrón Nro. 98271*
longo.gnr@hotmail.com

1er. Cuatrimestre de 2019
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

7/5/19

1. Enunciado

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 informe 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 6), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada caso.

4. Descripción
   En este trabajo, vamos a implementar dos programas en assembly MIPS, que nos permitan convertir información almacenada en streams de texto, de acuerdo a lo descripto en el trabajo anterior [1]: unix2dos, para transformar archivos de texto de UNIX a DOS; y dos2unix para realizar la operación inversa. Es decir, siguiendo la línea de trabajo de los prácticos anteriores, buscamos generar dos binarios diferentes en vez de un único programa que resuelva ambos problemas. Además, por tratarse de un TP orientado a practicar el conjunto de instrucciones, ambos programas deberán escritos completamente en assembly, en forma manual.

5. Interfaz
   Por defecto, ambos programas operan exclusivamente por stdin y stdout. Al igual que en el trabajo anterior, al finalizar, estos programas retornarán un valor nulo en caso de no detectar ningún problema; y, en caso contrario, finalizarán con código no nulo (por ejemplo 1).

6. Ejemplos
   Cuando el documento de entrada es vacío, la salida debería serlo también:

```
$ unix2dos </dev/null | wc -c
0
$ dos2unix </dev/null | wc -c
0
```

También se aplican todos los casos de prueba definidos para el TP anterior. Por ejemplo:

```
$ od -t c /tmp/dos.txt
0000000 U n o \r \n D o s \r \n T r e s \r \n
0000020
$ dos2unix -i /tmp/dos.txt -o - | od -t c
0000000 U n o \n D o s \n T r e s \n
0000015
```

Por último, podemos usar el programa que sigue para generar secuencias de datos aleatorias, pasarlas a hexa, reconvertirlas al dominio original, y verificar que la operación no haya alterado la información:

```
$ n=0; while :; do
> n="`expr $n + 1`";
> head -c 100 </dev/urandom >/tmp/test.$n.u;
> unix2dos </tmp/test.$n.u >/tmp/test.$n.d || break;
> dos2unix </tmp/test.$n.d >/tmp/test.$n.2.u || break;
> diff -q /tmp/test.$n.u /tmp/test.$n.2.u || break;
> rm -f /tmp/test.$n.*;
> echo Ok: $n;
> done; echo Error: $n
Ok: 1
Ok: 2
Ok: 3
...
```

7. Implementación
   El programa a implementar deberá satisfacer algunos requerimientos mínimos, que detallamos a continuación:

   a) ABI
   Será necesario que el código presentado utilice la ABI explicada en la clase: los argumentos correspondientes a los registros $a0-a3$ serán almacenados por el callee, siempre, en los 16 bytes dedicados de la sección "function call argument area" [2].

   b) Casos de prueba
   Es necesario que la implementación propuesta pase todos los casos incluidos tanto en este enunciado como en el conjunto pruebas suministrado en el informe del trabajo, los cuales deberán estar debidamente documentados y justificados.

   c) Documentación
   El informe deberá incluir una descripción detallada de las técnicas

y procesos de medición empleados, y de todos los pasos involucrados en el desarrollo del mismo, ya que forman parte de los objetivos principales del trabajo.

8. Informe
   El informe deberá incluir:
   - Este enunciado;
   - Documentación relevante al disenio, implementación y medición de las características del programa;
   - Las corridas de prueba, (sección 5.2) con los comentarios pertinentes;
   - El código fuente completo, en dos formatos: digitalizado1 e impreso en papel.
9. Fechas
   Fecha de vencimiento: 7/5.
10. Referencias Enunciado del primer trabajo práctico, primer cuatrimestre de 2019. http://groups.yahoo.com/groups/orga-comp/.

2. Implementación

Se implementaron dos funciones para poder convertir un archivo de DOS a UNIX y viceversa. La implementación se basó en un ciclo while que recorre el archivo byte por byte mientras consulta si está parado sobre un '\r' o '\n'. En caso de ser alguno de estos caracteres, se efectua la modificación correspondiente y sino, simplemente se escribe el mismo caracter en la salida. Para lograr esto, decidimos dividir los programas en pequeñas funciones.Esto nos permitió por un lado reducir la complejidad e ir teniendo seguridad de que las cosas que desarrollábamos funcionaban de la manera correcta. Y por otro lado, reutilizar funciones que aplicaban tanto a unix2dos como a dos2unix. Las funciones desarroladas fueron:

- getch: devuelve el caracter leído por stdin y en caso de ser EOF devuelve -1.
- putch: escribe el caracter que recibe por stdout.
- isLF: devuelve 1 si el caracter es LF y 0 si no lo es.
- isCR: devuelve 1 si el caracter es CR y 0 si no lo es.

3. Utilización

4. Generación de binarios

5. Pruebas

6. Código fuente

6.1. getch

```
#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl getch
.ent getch

getch:
.frame $fp, 40, ra
.set noreorder
.cpload t9
.set reorder
```

```
subu sp, sp, 40  # pido espacio para mi Stack Frame
.cprestore 28 # salvo gp en 28
sw $fp, 24(sp) # salvo fp en 24
sw ra, 32(sp) # salvo ra en 32
move $fp, sp # a partir de acá trabajo con fp

# me guardo los parámetros que no guardo la caller (por convención de ABI)
sw a0, 40($fp) # salvo el file descriptor
sw a1, 44($fp) # salvo el buffer adonde va lo leído (en este caso es un int)
sw a2, 48($fp) # salvo la cantidad de bytes a leer (es 1)

# Se supone que el file descriptor es STDIN (0) y la cantidad de bytes es 1
# hago la lectura propiamente dicha:
_lectura:
la a0, 0 # STDIN
lw a1, 44($fp)  # buffer de lo leido
li a2, 1 # (1 acá es 1 byte)
li v0, SYS_read
syscall
#move a1, v1

_comprobacion:
bltz a3, _error
beqz v0, _eof # (en a3 ya tengo un 0) si en v0 tengo un 0, entonces EOF
li t0, 1
beq t0, v0, _return # esto ya es obvio, en realidad

_return:
# tengo en v0 la posición en memoria del char (leído) y
# me lo llevo con lb -> mucho muy importante, porque es un char
lb v0, 44($fp) # en v0 guardo el char (leído) que es lo que voy a devolver
(o -1 si terminó)
# Devolvemos las cosas a su lugar
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
jr ra

_eof:
li t0, -1
sw t0, 44($fp)
b _return
```

```
_error: # en caso de error, aborto el programa con código 1
li a0, 1
li v0 SYS_exit
syscall


.end getch
```

6.2. putch

```
#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl putch
.ent putch

putch:
.frame $fp, 40, ra
.set noreorder
.cpload t9
.set reorder

subu sp, sp, 40  # pido espacio para mi Stack Frame
.cprestore 28 # guardo gp en 28
sw $fp, 24(sp) # guardo fp en 24
sw ra, 32(sp) # guardo ra en 32
move $fp, sp # a partir de acá trabajo con fp

# me guardo los parámetros que no guardo la caller (por convención de ABI)
# EL CHAR QUE LLEGA A ÉSTA ALTURA TIENE QUE HABER SIDO CARGADO CON lb !!!!!!!
sw a0, 40($fp) # guardo FD (que debería ser 1, stdout)
sw a1, 44($fp) # guardo el char a escribir
sw a2, 48($fp) # guardo size a escribir (va a ser 1)

_escritura:
la a0, 1 # stdout
lw t0, 44($fp)
la a1, 0(t0)
li a2, 1 # cantidad de bytes
li v0, SYS_write
syscall

bnez a3, _escritura  # Si da error intentamos de nuevo
```

```
_return:
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
jr ra

.end putch
```

6.3. isLF

```
#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl isLF
.ent isLF

isLF:
.frame $fp, 40, ra
.set noreorder
.cpload t9
.set reorder

subu sp, sp, 40  # pido espacio para mi Stack Frame
.cprestore 28 # guardo gp en 20
sw $fp, 24(sp) # guardo fp en 16
sw ra, 32(sp) # guardo ra en 24
move $fp, sp # a partir de acá trabajo con fp

sw a0, 40($fp) # me guardo el char a comparar a 40 de fp, aunque sea innecesario
lb t1, 0(a0) # cargo el char en sí ES UN BYTE
la t0, LF
lb t2, 0(t0) # cargo el valor de LF según código ASCII

beq t1, t2, _true
b _false

_true:
li t0, 1
sb t0, 16($fp)
b _end
```

```
_false:
li v0, 0
sb t0, 16($fp)

_end:
# cargo en v0 el vaLor del booleano
lb v0, 16($fp)
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
jr ra

.end isLF

.data

LF: .byte 10
```

6.4. isCR

```
#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl isCR
.ent isCR

isCR:
.frame $fp, 40, ra
.set noreorder
.cpload t9
.set reorder

subu sp, sp, 40  # pido espacio para mi Stack Frame
.cprestore 28 # guardo gp en 20
sw $fp, 24(sp) # guardo fp en 16
sw ra, 32(sp) # guardo ra en 24
move $fp, sp # a partir de acá trabajo con fp

sw a0, 40($fp) # me guardo el char a comparar a 40 de fp, aunque sea innecesario
lb t1, 0(a0) # cargo el char en sí ES UN BYTE
la t0, CR
lb t2, 0(t0) # cargo el valor de CR según código ASCII
```

```
beq t1, t2, _true
b _false

_true:
li t0, 1
sb t0, 16($fp)
b _end

_false:
li v0, 0
sb t0, 16($fp)

_end:
# cargo en v0 el vaLor del booleano
lb v0, 16($fp)
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
jr ra

.end isCR

.data

CR: .byte 13
```

6.5. unix2dos

```
#include <mips/regdef.h>
#include <sys/syscall.h>
.text
.abicalls
.globl main
.ent main

main:
.frame $fp, 40, ra
.set noreorder
.cpload t9
.set reorder

subu sp, sp, 40  # pido espacio para mi Stack Frame
```

```
.cprestore 28 # guardo gp en 28
sw $fp, 24(sp) # guardo fp en 24
sw ra, 32(sp) # guardo ra en 32
move $fp, sp # a partir de acá trabajo con fp

# hago una lectura:
_lectura:

la t0, STDIN
lw a0, 0(t0) # STDIN
la a1, BUFFER
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal getch

# en v0 me devuelve el byte leído
li t0, -1
beq v0, t0, _eof

# me fijo si tengo un LF
la a0, BUFFER
jal isLF

li t0, 1
beq v0, t0, _replace_LF

la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, BUFFER
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch

b _lectura

_eof:
_return:
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
addu sp, sp, 40
li v0, SYS_exit
li a0, 0
syscall
```

```
_replace_LF:
la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, CR
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch

la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, BUFFER # acá sigo teniendo LF
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch

b _lectura

.end main

.data
BUFFER: .space 1
CR: .byte 13
STDIN: .word 0
STDOUT: .word 1
BYTE: .byte 1
```

6.6. dos2unix

```
#include <mips/regdef.h>
#include <sys/syscall.h>
.text #Lo que viene es codigo
.abicalls #Voy a usar la convencion abi
.globl main
.ent main


main:
.frame $fp, 40, ra
.set noreorder #RECETA magica no explicada
.cpload t9 #RECETA magica no explicada
.set reorder #RECETA magica no explicada

subu sp, sp, 40  # pido espacio para mi Stack Frame
```

```
.cprestore 28 # guardo gp en 28
sw $fp, 24(sp) # guardo fp en 24
sw ra, 32(sp) # guardo ra en 32
move $fp, sp # a partir de acá trabajo con fp

# hago una lectura:
_lectura:

# Cargo el primer argumento
la t0, STDIN #Cargo STDIN(0) en el registro t0
lw a0, 0(t0) #Cargo t0 en el registro a0
# Cargo el segundo argumento
la a1, BUFFER
# Cargo el tercer argumento
la t0, BYTE
lw a2, 0(t0) # 1 byte

# Llamo a getch
jal getch # en v0 me devuelve el byte leído

li t0, -1
beq v0, t0, _eof

# me fijo si tengo un CR
la a0, BUFFER
jal isCR

li t0, 1
beq v0, t0, _checkNextIsEOF

la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, BUFFER
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch

b _lectura

_eof:
_return:
lw ra, 32(sp)
lw $fp, 24(sp)
lw gp, 28(sp)
```

```
addu sp, sp, 40
li a0, 0
li v0, SYS_exit
syscall

_checkNextIsEOF:
# Llamo a getch
jal getch # en v0 me devuelve el byte leído


li t0, -1
beq v0, t0, _putCRandBreak


# me fijo si tengo un LF
la a0, BUFFER
jal isLF

li t0, 1
beq v0, t0, _putLF

la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, CR
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch

la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, BUFFER
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch

b _lectura

_putCRandBreak:
la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, CR
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch
```

```
b _eof # break;

_putLF:
la t0, STDOUT
lw a0, 0(t0) # STDOUT
la a1, LF
la t0, BYTE
lw a2, 0(t0) # 1 byte
jal putch

b _lectura

.end main

.data
BUFFER: .space 1
CR: .byte 13
LF: .byte 10
STDIN: .word 0
STDOUT: .word 1
BYTE: .byte 1
```

7. Conclusiones