Trabajo Práctico N° 1: Conjunto de Instrucciones MIPS

Bosco, Mateo - *Padrón Nro. 93.488*
mateo.bosco@hotmail.com

Hidalgo, Juan Manuel - *Padrón Nro. 93.383*
juanmah_@hotmail.com

2do. Cuatrimestre de 2013
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

1. Introducción

El siguiente informe corresponde a la documentación de la aplicación tp1, detallando los pasos necesarios para compilar y ejecutar dicha aplicación y para la correcta utilización de la misma. La portabilidad de tp1 se asegura para los sistemas operativos Linux (Ubuntu) y NetBSD.

El objetivo del presente trabajo ha sido realizar una versión del comando *rev* de UNIX escrito en lenguaje C con la implementacion de la funcion *reverse()* en codigo MIPS. El mismo concatena y escribe en stdout el contenido de uno o más archivos, invirtiendo el orden de los caracteres de cada línea.

En las siguientes secciones se detalla como se ha implementado el mismo, así como las correspondientes corridas de prueba y su código fuente en lenguaje C y el código MIPS32.

2. Diseño, Implementación y Consideraciones Generales

Para la implementación del programa utilizamos la estructura de pila en donde fuimos apilando letras para luego desapilarlas. Comenzamos recorriendo lo introducido en **stdin** guardándolo en un buffer en memoria dinámica para luego ir colocando en la pila letra por letra hasta que se encuentre un enter, es decir que termina el renglón, o hasta que sea el fin del **stdin**. Luego se desapilan las letras colocándolas en un vector para luego escribirlo en **stdout**, de esta forma queda invertida cada línea.

Para la implementacion del trabajo utilizamos varios archivos:

- **main.c**: este programa escrito en C se encarga del manejo de los archivos, el procesamiento de las opciones por linea de comando y el reporte de errores.
- **reverse.S**: este programa escrito en codigo MIPS contendra la implementacion de la funcion *reverse()* que se encarga de leer una linea y dar vuelta sus caracteres y tambien contendra la definicion del vector *reverse_errmsg[]* que sera equivalente a un vector C como *const char* reverse_errmsg[]*. Este vector sera de 5 posiciones, la primera para la correcta finalizacion del programa y las 4 restantes indicando donde se ocurrio el error durante la ejecucion.
- **reverse.h**: este sera el archivo *header* donde se incluira la declaracion de *reverse()* para que se la pueda llamar desde main.c

Para la implementacion de la funcion *reverse()* en codigo MIPS, se dividio la tarea en 3 funciones distintas:

- *swap_enlinea*: se encarga de invertir el orden de aparicion de los caracteres dentro de una linea, esta funcion hace el intercambio in situ, es decir, no devuelve otro arreglo sino que cambia los caracteres en el lugar. De este modo, el que era el ultimo caracter del arreglo pasa a ser el primero, el que era el primer pasa a ser el ultimo y asi con todos los caracteres del arreglo. Se recibe como parametro un puntero al arreglo y el largo de este, devuelve el mismo arreglo invertido.
- *lineaLeer*: esta funcion recibe como parametro el fd del archivo a leer, llama a *mymalloc* para pedir memoria para un vector con un tamano definido como constante (*TAM_CADENA*) y luego va copiando los caracteres del archivo dentro del vector hasta encontrar un '\n' o el EOF y comprobando que el tamano de la linea siempre sea menor que el vector, en caso contrario llamando nuevamente a *mymalloc* para pedir un nuevo buffer que sea el doble de grande, copia el contenido y luego a llama *myfree* para liberar el buffer viejo. Una vez terminado, devuelve el puntero al vector con la linea leida
- *reverse*: esta es la funcion que se encarga de llamar a todas las otras. Primero obtiene por parametro los dos fd de los archivos de entrada y salida, luego llama a *lineaLeer* y con el puntero y el largo que esta funcion le devuelve, llama a *swap_enlinea* para dar vuelta la linea. Una vez terminado esto, escrive el vector invertido en el archivo de salida.

3. Utilización del programa

3.1. -h ó -help

Se puede obtener información al instante del programa ejecutando: **./tp1 -h** o bien **./tp1 -help** obteniéndose:

```
$ tp1 -h
Usage:
  tp1 -h
  tp1 -V
  tp1 [file...]
Options:
  -V, --version         Print version and quit.
```

```
  -h, --help            Print this information and quit.
Examples:
  tp1 foo.txt bar.txt
  tp1 gz.txt
  tp1 <stdin>
```

3.2. -V ó -version

Se puede obtener información acerca de la versión del programa ejecutando ./**tp1 -V** o bien **./tp1 -version** obteniéndose:

```
tp1 - 1
(66.20 Organizacion de Computadoras - FIUBA)
2do cuatrimestre 2013.
Escrito por  Mateo Bosco y Juan Manuel Hidalgo.
```

4. Generación del archivo binario

Para compilar el programa se deben seguir los siguientes pasos:

1. Copiar los archivos que se encuentran dentro de la carpeta tp1 a un directorio dentro de NetBSD como fue explicado en clase
2. Compilar con los siguientes comandos desde la terminal de GxEmul

   ```
   gcc -c reverse.S
   gcc -c ./sys_mmap/mymalloc.S -o ./mymalloc.o
   gcc mymalloc.o reverse.o main.c -o tp1
   ```

3. Ejecutar el tp1, por ejemplo con el comando desde la terminal de GxEmul

   ```
   ./tp1 prueba.txt
   ```

5. Ejecución de la aplicación

Para correr tp1, situarse en el directorio donde se encuentra el binario creado en el paso anterior (tp1) y escribir: ./tp1 -h
Esto mostrará la ayuda, con las distintas formas de uso de tp1.
Todos los archivos de prueba usados en esta sección y la siguiente se encuentran en el CD adjunto a este trabajo.

5.1. Ejemplos

```
$ cat prueba1.in
Se agrava la situacion en Cordoba
 por los incendios y ya hay cerca d
e 500 evacuados.

$ cat prueba1.in | ./tp1
abodroC ne noicautis al avarga eS
d acrec yah ay y soidnecni sol rop
.sodaucave 005 e

$ cat prueba2.in
El aleman Thomas Bach sera quien este al frente del
Comite Olimpico Internacional (COI) en los proximos ocho anos.
Era el favorito antes de la eleccion, y sucedera al belga Jacques Rogge,
quien fue presidente desde 2001.

$ ./tp1 prueba2.in
led etnerf la etse neiuq ares hcaB samohT namela lE
.sona ohco somixorp sol ne )IOC( lanoicanretnI ocipmilO etimoC
,eggoR seuqcaJ agleb la aredecus y ,noiccele al ed setna otirovaf le arE
.1002 edsed etnediserp euf neiuq
```

6. Corridas de Prueba

6.1. Archivo de Entrada prueba.txt

```
Listado de paises por producto interno bruto (PIB) a precios nominales:

Posicion        Pais            PBI (millones de USD)
1               Estados Unidos  15.653.366
2               China           8.250.241
25              Argentina       474.812
```

6.2. tp1 desde la línea de comandos

```
$ ./tp1 prueba.txt
:selanimon soicerp a )BIP( oturb onretni otcudorp rop sesiap ed odatsiL

)DSU ed senollim( IBP           siaP    noicisoP
663.356.51      sodinU sodatsE          1
142.052.8               anihC           2
218.474 anitnegrA               52
```

6.3. Prueba desde entrada estandar (stdin)

```
$ ./tp1
Hola, que tal?
?lat euq ,aloH
esto es una prueba.
.abeurp anu se otse
```

6.4. Prueba de prueba1.in y prueba2.in simultánea y comparación con comando rev

```
$ ./tp1 prueba1.in prueba2.in
abodroC ne noicautis al avarga eS
d acrec yah ay y soidnecni sol rop
.sodaucave 005 e
led etnerf la etse neiuq ares hcaB samohT namela lE
.sona ohco somixorp sol ne )IOC( lanoicanretnI ocipmilO etimoC
,eggoR seuqcaJ agleb la aredecus y ,noiccele al ed setna otirovaf le arE
.1002 edsed etnediserp euf neiuq

$ rev prueba1.in prueba2.in
abodroC ne noicautis al avarga eS
d acrec yah ay y soidnecni sol rop
.sodaucave 005 e
led etnerf la etse neiuq ares hcaB samohT namela lE
.sona ohco somixorp sol ne )IOC( lanoicanretnI ocipmilO etimoC
,eggoR seuqcaJ agleb la aredecus y ,noiccele al ed setna otirovaf le arE
.1002 edsed etnediserp euf neiuq
```

6.5. Comparación de las salidas de tp1 y rev utilizando el comando cat y el comando od

```
$ cat prueba1.in | ./tp1 | od -t x1
0000000 61 62 6f 64 72 6f 43 20 6e 65 20 6e 6f 69 63 61
0000020 75 74 69 73 20 61 6c 20 61 76 61 72 67 61 20 65
0000040 53 0a 64 20 61 63 72 65 63 20 79 61 68 20 61 79
0000060 20 79 20 73 6f 69 64 6e 65 63 6e 69 20 73 6f 6c
0000100 20 72 6f 70 20 0a 2e 73 6f 64 61 75 63 61 76 65
0000120 20 30 30 35 20 65 0a
0000127

$ cat prueba1.in | rev | od -t x1
0000000 61 62 6f 64 72 6f 43 20 6e 65 20 6e 6f 69 63 61
```

```
0000020 75 74 69 73 20 61 6c 20 61 76 61 72 67 61 20 65
0000040 53 0a 64 20 61 63 72 65 63 20 79 61 68 20 61 79
0000060 20 79 20 73 6f 69 64 6e 65 63 6e 69 20 73 6f 6c
0000100 20 72 6f 70 20 0a 2e 73 6f 64 61 75 63 61 76 65
0000120 20 30 30 35 20 65 0a
0000127
```

6.6. Comparación de las salidas de tp1 y rev utilizando el comando hexdump

```
$ ./tp1 prueba2.in | hexdump -c
0000000   l   e   d       e   t   n   e   r   f       l   a       e   t
0000010   s   e       n   e   i   u   q       a   r   e   s       h   c
0000020   a   B       s   a   m   o   h   T       n   a   m   e   l   a
0000030       l   E  \n   .   s   o   n   a       o   h   c   o       s
0000040   o   m   i   x   o   r   p       s   o   l       n   e       )
0000050   I   O   C   (       l   a   n   o   i   c   a   n   r   e   t
0000060   n   I       o   c   i   p   m   i   l   O       e   t   i   m
0000070   o   C  \n   ,   e   g   g   o   R       s   e   u   q   c   a
0000080   J       a   g   l   e   b       l   a       a   r   e   d   e
0000090   c   u   s       y       ,   n   o   i   c   c   e   l   e
00000a0   a   l       e   d       s   e   t   n   a       o   t   i   r
00000b0   o   v   a   f       l   e       a   r   E  \n   .   1   0   0
00000c0   2       e   d   s   e   d       e   t   n   e   d   i   s   e
00000d0   r   p       e   u   f       n   e   i   u   q  \n
00000dd

$ rev prueba2.in | hexdump -c
0000000   l   e   d       e   t   n   e   r   f       l   a       e   t
0000010   s   e       n   e   i   u   q       a   r   e   s       h   c
0000020   a   B       s   a   m   o   h   T       n   a   m   e   l   a
0000030       l   E  \n   .   s   o   n   a       o   h   c   o       s
0000040   o   m   i   x   o   r   p       s   o   l       n   e       )
0000050   I   O   C   (       l   a   n   o   i   c   a   n   r   e   t
0000060   n   I       o   c   i   p   m   i   l   O       e   t   i   m
0000070   o   C  \n   ,   e   g   g   o   R       s   e   u   q   c   a
0000080   J       a   g   l   e   b       l   a       a   r   e   d   e
0000090   c   u   s       y       ,   n   o   i   c   c   e   l   e
00000a0   a   l       e   d       s   e   t   n   a       o   t   i   r
00000b0   o   v   a   f       l   e       a   r   E  \n   .   1   0   0
00000c0   2       e   d   s   e   d       e   t   n   e   d   i   s   e
00000d0   r   p       e   u   f       n   e   i   u   q  \n
00000dd
```

6.7. Comparación de las salidas de tp1 y rev utilizando el comando diff

```
$ ./tp1 prueba1.in > temp1
$ rev prueba1.in > temp2
$ diff -s temp1 temp2
Files temp1 and temp2 are identical

$ ./tp1 prueba2.in > temp1
$ rev prueba2.in > temp2
$ diff -s temp1 temp2
Files temp1 and temp2 are identical

$ ./tp1 prueba.txt > temp1
$ rev prueba.txt > temp2
$ diff -s temp1 temp2
Files temp1 and temp2 are identical
```

7. Código Fuente de main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "reverse.h"


int main(int argc, char** argv){
        int numero_archivos = argc - 1;
        int archivo;
        bool noFile = false;
        if (numero_archivos == 0){
                archivo = 0;
                numero_archivos = 1;
                noFile = true;
        }
        else if (strcmp(argv[1],"-h")==0 && (numero_archivos==1)){
                printf("Usage:\ntp1 -h\ntp1 -V\ntp1 [file ...]\nOptions:\n-V, --version  
                    Print version and quit.\n-h, --help  Print this information and quit.\
                    nExamples:\nt10 foo.txt bar.txt\ntp1 gz.txt\ntp1 <stdin> \n");
                return 0;
                }
        else if (strcmp(argv[1],"-V")==0 && (numero_archivos==1)){
                printf("tp1 - 1 \n(66.20 Organizacion de Computadoras - FIUBA)\n2do 
                    cuatrimestre 2013.\nEscrito por  Mateo Bosco y Juan Manuel Hidalgo. \n
                    ");
                return 0;
                }
        int i = 0;
        int resultado = 0;
        while (i < numero_archivos){
                if (! noFile){
                        archivo = open(argv[i+1],O_RDONLY);
                }

                if(archivo < 0){ // file < 0 es error
                        fprintf(stderr,"Ocurrio un error al abrir el archivo %s\n. El 
                            programa terminara ahora.",argv[i+1]);
                        exit(1);
                } else {
                                                resultado = reverse(archivo, 1);
                        close(archivo);
                        if(resultado != 0){
                                printf("%s\n",reverse_errmsg[resultado]);
                                return resultado;
                        }
                }
                i++;
        }
        printf("%s\n", reverse_errmsg[resultado]);
        return 0;
}
```

8. Código Fuente de reverse.h

```
#ifndef REVERSE_H
#define REVERSE_H

#include <stdio.h>
#include "sys_mmap/mymalloc.h"

extern const char* reverse_errmsg[];

int reverse(int infd, int outfd);

char* lineaLeer(int fd,int* length);

#endif
```

9. Código MIPS de reverse.S

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


#define TAM_CADENA 30


.text
.align 2
.globl reverse
.ent reverse
reverse:
        .frame $fp , 40 , ra
    .set noreorder
    .cpload t9
    .set reorder
    .cprestore 16

        subu sp , sp , 40                       #RESERVO UN LUGAR PARA LA PILA DE 40
        li t0 , 10                                      # '\n' = 10, lo cargo en t0
        sb t0 , 20 (sp)                         #guardo el '\n' en la pila
        sw $fp , 32 (sp)                        #guardo el fp
        sw gp , 36 (sp)                         #guardo el gp
        sw ra , 40 (sp)                         #guardo el ra
        sw a0 , 44 (sp)                         #guardo fd_in porque a0 se puede a
            modificar
        sw a1 , 48 (sp)                         #guardo fd_out porque a1 se puede a
            modificar


while:
                #LLAMADA A LINEALEER
        li t0 , 0                                       #t0 es el tamano inicializado en 0
        lw a0 , 44 (sp)                         #cargo a0
        sw t0 , 24 (sp)                         #guardo el largo en la pila
        addiu a1 , sp , 24                      #cargo en a1 la direccion de memoria donde
             se encuentra el largo en la pila, parametro de LineaLeer
```

```
        jal lineaLeer                           #llamo a la funcion
        lw t0 , 24 (sp)                         #cargo el largo de la linea

        bnez v1 , error                         #compruebo errores
        beqz t0 , salida                        #verifico que size sea mayor o igual a 0

                #LLAMADA A SWAP_ENLINEA
        move a0 , v0                            #paso por parametro a0 el puntero a la
           linea
        move a1 , t0                            #paso por parametro a1 el tamano
        jal swap_enlinea                        #llamo a la funcion swap_enlinea

                #ESCRITURA
        move a1 , a0                            #pongo en a1 la linea leida
        li v0 , SYS_write
        lw a0 , 48(sp)                          #cargo en a0 el fd_out
        lw a2 , 24 (sp)                         #cargo en a2 el largo
        syscall
        blez v0 , manejo_error_escritura

                #FREE
        move a0 , a1                            #cargo en a0 la linea que fue leida
        jal myfree                                      #llamo a myfree para liberar la
           linea
        bnez v0, manejo_error_myfree2

                #ESCRITURA \n
        addu a1 , sp , 20                       #cargo en a1 el puntero a la direccion del
            '\n' guardado en la pila
        li v0 , SYS_write
        li a0 , 1
        li a2 , 1
        syscall
        blez v0 , manejo_error_escritura

        j while

manejo_error_myfree2:
        li v1 , 4                                       #codigo de error en 4
        b error
manejo_error_escritura:
        li v1 , 2                                       #codigo de error en 2

error:
move v0,v1                                              #devuelvo el codigo de error
b desapilar

salida:
        li v0 , 0                                       #devuelvo 0 en v0, se ejecuto sin
           problemas

desapilar:
        lw $fp , 32 (sp)
        lw gp , 36 (sp)
        lw ra , 40 (sp)
        lw a0 , 44 (sp)
        lw a1 , 48 (sp)
        addiu sp , sp , 40
        jr ra

.end reverse
```

```
.ent swap_enlinea
swap_enlinea:
        subu sp , sp, 8                         #solo hay que guardar fp y gp
        sw $fp , 0(sp)
        sw gp , 4(sp)
        sw a0 , 12(sp)
        sw a1 , 16(sp)
        move t0 , a0                            #t0 puntero a vector
        move t1 , a1                            #t1 largo del vector
        beqz t0 , fin_swap
        li t2 , 0                                       #cargo 0 en t2
        addiu t1 , t1 , -1                      #largo--

loop:                                                           #i=t2 j=t1
        bgt t2 , t1 , fin_swap          #si t1>t2 salto a fin_swap
        addu t5 , t0 , t2                       #cargo t0+t1 en t5
        lb t3 , 0 (t5)                          #cargo en t3 el el valor de vector[i]
        addu t6 , t0 , t1                       #cargo t0+t2 en t6
        lb t4 , 0 (t6)                          #cargo en t4 el valor de vector[l]
        sb t4 , 0 (t5)                          #swap1
        sb t3 , 0 (t6)                          #swap2
        addiu t2 , t2 , 1                       #i++
        addiu t1 , t1 , -1                      #l--
        b loop                                          #salto a loop

fin_swap:
        lw $fp , 0 (sp)
        lw gp , 4 (sp)
        lw a0 , 12 (sp)
        lw a1 , 16 (sp)
        addiu sp , sp , 8
        jr ra

.end swap_enlinea



.globl lineaLeer
.ent lineaLeer
lineaLeer:
        .frame $fp , 40 , ra
    .set noreorder
    .cpload t9
    .set reorder
    .cprestore 16

        subu sp , sp , 40
        sw $fp , 12 (sp)
        sw gp , 16 (sp)
        sw ra , 20 (sp)
        sw s0 , 24 (sp)
        sw s1 , 28 (sp)
        sw s2 , 32 (sp)
        sw s3 , 36 (sp)
        sw s4 , 40 (sp)
        sw a0 , 44 (sp)                                 #guardo el fd del archivo
        sw a1 , 48 (sp)                                 #guardo el puntero al largo del
            archivo
```

```
        move s0 , a0                                    #muevo el fd al s0
        move s1 , a1                                    #muevo el puntero al largo a s1
        move s2 , zero                                  #cargo cero en i

                #MALLOC BUFFER
        li a0 , TAM_CADENA                              #resevo memoria para la linea
        move t7 , a0
        jal mymalloc                                    #llamo a mymalloc para reservar
            memoria para la linea
        move s3 , v0                                    #muevo a s3 el valor que devuelve
            mymalloc
        beqz s3 , manejo_error_mymalloc #manejo de errores de mymalloc

                #MALLOC CARACTER
        li a0 , 1                                               #reservo memoria para un
            caracter
        jal mymalloc                                    #llamo a mymalloc para reservar
            memoria para un caracter
        move s4 , v0                                    #muevo a s4 el valor que devuelve
            mymalloc
        beqz s4 , manejo_error_mymalloc #manejo de errores de mymalloc


        li s7 , TAM_CADENA                              #cargo en s7 el tamano inicial de
            la linea

ciclo:
        li v0 , SYS_read                                #SYScall de read para leer un
            caracter
        move a0 , s0                                    #paso el primer parametro, fd al
            archivo
        move a1 , s4                                    #paso el segundo parametro,
            puntero al mymalloc del caracter
        li a2 , 1                                               #paso el tercer parametro,
             cantidad de caracteres a leer
        syscall
        bltz v0 , manejo_error_sysread

        lb t1 , 0 (s4)                                  #cargo en t1 la letra
        addu t2 , s3 , s2                               #cargo en t2 el valor de linea[i]
        sb t1 , 0 (t2)                                  #cargo en linea[i] el valor de la
            letra
        addi s2 , 1                                             #incremento i

        bne s7 , s2 , label

                #INCREMENTAR BUFFER

        move t3 , s7                                            #cargo en t2 el tamano del
             buffer1
        addu s5 , t3 , t3                                       #TAMANO CADENA *=2 se
            guarda en s5
        move a0 , s5                                            #pongo el a0 el parametro
            tamano para mymalloc
        move t7 , a0
        jal mymalloc                                            #llamo a mymalloc para
            reallocar
        move s6 , v0                                            #guardo en s6 el buffer2
        beqz s6 , manejo_error_mymalloc         #manejo de errores de mymalloc
        move t2 , zero                                          #t2 indice k para realloc
```

```
copiar:
        addu t4 , s3 , t2                                   #t4 = buffer1+k
        addu t5 , s6 , t2                                   #t5 = buffer2+k
        lb t6 , 0 (t4)                                      #t6 = buffer1[k]
        sb t6 , 0 (t5)                                      #buffer2[k] = buffer1[k]
        addi t2 , 1                                                 #k++
        blt t2 , s7 , copiar                            #si k<largo1 llamo a copiar
        move a0 , s3                                        #paso por parametro a0 el
           puntero al buffer1
        jal myfree                                                  #llamo a myfree
           para liberar el buffer1
        bnez v0 , manejo_error_myfree           #manejo el error de myfree
        move s3 , s6                                        #buffer1 = buffer2
        move s7 , s5                                        #largo1 = largo2

                #FIN INCREMENTAR BUFFER

label:
        li t2 , 10                                          #cargo el t2 el valor de
           '\n'
        lw t1 , 0 (s4)                                  #cargo en 21 el valor de EOF
        beq t1 , t2 , fin_ciclo                 #si la letra es un enter
        beqz t1 , fin_ciclo                             #si la letra es un EOF
        b ciclo

fin_ciclo:
        addi s2 , -1                                    #resto 1 a i para escrbir arriba
           del enter o el EOF
        addu t2 , s3 , s2                               #cargo en linea[i] el valor de t2
        li t1 , 0                                           #cargo un cero en t1
        sb t1 , 0 (t2)                                  #cargo en linea[i] un '\0' para
           indicar el final de una cadena

                #FREE
        move a0 , s4                                    #paso por parametro en a0 el
           puntero a la memoria que devuelve mymalloc
        jal myfree                                          #llamo a myfree para
           liberar memoria
        bnez v0 , manejo_error_myfree   #manejo el error de myfree

        bnez s2 , buen_fin                              #llamo a buen_fin
        move a0 , s3                                    #si el largo es 0 libero memoria
        jal myfree                                          #llamo a myfree
        bnez v0 , manejo_error_myfree   #manejo error de myfree
        b manejo_final                          #llamo a manejo_final

manejo_error_sysread:
        li v1 , 1                                           # codigo de error en 1
        b final

manejo_error_mymalloc:
        li v1 , 3                                           # codigo de error en 3
        b final

manejo_error_myfree:
        li v1 , 4                                           # codigo de error en 4
        b final

manejo_final:
        li v1 , 0                                           # codigo de error en 0
```

```
final:
        move v0 , zero
        sw zero , 0 (s1)                                #paso el largo como parametro
        b fin

buen_fin:
        move v0 , s3                                    #devuelvo el puntero a la linea
        li v1 , 0                                               #codigo de error en 0
        sw s2 , 0 (s1)                                  #paso el largo de parametro
        b fin

fin:
        lw $fp , 12 (sp)
        lw gp , 16 (sp)
        lw ra , 20 (sp)
        lw s0 , 24 (sp)
        lw s1 , 28 (sp)
        lw s2 , 32 (sp)
        lw s3 , 36 (sp)
        lw s4 , 40 (sp)
        lw a0 , 44 (sp)
        lw a1 , 48 (sp)
        addiu sp, sp, 40
        jr ra
.end lineaLeer

.data
.align 2
.globl reverse_errmsg                   #VECTOR DE ERRORES
reverse_errmsg: .word bien, lectura, escritura, malloc, free
        .size reverse_errmsg, 16

.align 0

bien: .asciiz "Se ejecuto sin problemas"        #CODIGO DE ERROR EN 0
lectura: .asciiz "error de lectura"                     #CODIGO DE ERROR EN 1
escritura: .asciiz "error de escritura"         #CODIGO DE ERROR EN 2
malloc: .asciiz "error en mymalloc"                     #CODIGO DE ERROR EN 3
free: .asciiz "error en myfree"                         #CODIGO DE ERROR EN 4
```

10. Conclusiones

Con este trabajo pudimos observar que es posible migrar la logica de un programa escrito en C a codigo Assembly. Si bien esto implica un trabajo extra y unas cuantas horas mas de programacion, este trabajo nos podria ser muy util para programas donde la velocidad de procesamiento es importante. Por ejemplo, para algoritmos de ordenamiento seria una buena mejora programar la logica del ordenamiento en Assembly mientras que el manejo de archivos y errores lo puede manejar tranquilamente un programa escrito en un lenguaje de mas alto nivel como C

Universidad de Buenos Aires, F.I.U.B.A.
66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
2^{do} cuatrimestre de 2013

`$Date: 2013/11/07 02:34:04 $`

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, se reimplementará parcialmente en assembly MIPS el programa desarrollado en el trabajo práctico anterior [1].

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

- `main.c`: contendrá todo el código necesario para el procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores (`stderr`). Desde aquí se llama a la función que invierte líneas siguiente.

- `reverse.S`: contendrá el código MIPS32 assembly con la función `reverse()`, y las funciones y estructuras de datos auxiliares que los alumnos crean convenientes (ej: para reserva de memoria). También contendrá la definición en assembly de un vector equivalente al siguiente vector C: `const char* reverse_errmsg[]`. Dicho vector contendrá los mensajes de error que las funciones antes mencionadas puedan generar, y cuyo índice es el código de error devuelto por las mismas.

- Los header files pertinentes (al menos, `reverse.h`, con el prototipo de reverse(), a incluir en `main.c`) y la declaración del vector `extern const char* reverse_errmsg[]`[1].

A su vez, el prototipo C de la función MIPS32 `reverse()` es el siguiente:

- `int reverse(int infd, int outfd)`

La función recibe por `infd` y `outfd` los file descriptors correspondientes a los archivos de entrada y salida pre-abiertos por `main.c`.

Ante un error, ambas funciones volverán con un código de error numérico (índice del vector de mensajes de error de `reverse.h`), o cero en caso de realizar el procesamiento de forma exitosa.

5. Implementación

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

5.1. ABI

Será necesario que el código presentado utilice la ABI explicada en clase ([2] y [3]).

5.2. Syscalls

Es importante aclarar que desde el código assembly no podrán llamarse funciones que no fueran escritas originalmente en assembly por los alumnos (o las provistas por la cátedra). Por lo contrario, desde el código C sí podrá (y deberá) invocarse código assembly.

[1] no confundir con la definición, que deberá implementarse en assembly dentro de reverse.S

Por ende, y atendiendo a lo planteado en la sección 4, los alumnos deberán invocar algunos de los system calls disponibles en NetBSD (en particular, `SYS_read` y `SYS_write`).

5.3. Casos de prueba

Es necesario que la implementación propuesta pase todos los casos incluidos tanto en el enunciado del trabajo anterior [1] como en el conjunto de pruebas suministrado en el informe del trabajo, los cuales deberán estar debidamente documentados y justificados.

5.4. Documentación

El informe deberá incluir una descripción detallada de las técnicas y procesos de desarrollo y debugging empleados, ya que forman parte de los objetivos principales del trabajo.

6. Informe

El informe deberá incluir:

- Este enunciado;
- Documentación relevante al diseño, desarrollo y debugging del programa;
- Las corridas de prueba, (sección 5.3) con los comentarios pertinentes;
- El código fuente completo, en dos formatos: digitalizado[2] e impreso en papel.

7. Fechas

La fecha de la primera oportunidad de entrega, es el martes 19/11. La fecha de vencimiento (y fin del curso) es el martes 3/12.

Referencias

[1] Enunciado del primer trabajo práctico (TP0), segundo cuatrimestre de 2013 (`http://groups.yahoo.com/groups/orga-comp/files/TPs/`).

[2] System V application binary interface, MIPS RISC processor supplement (third edition). Santa Cruz Operations, Inc.

[2] No usar diskettes: son propensos a fallar, y no todas las máquinas que vamos a usar en la corrección tienen lectora. En todo caso, consultá con tu ayudante.

[3] MIPS ABI: Function Calling Convention, Organización de computadoras - 66.20 (archivo "func_call_conv.pdf", `http://groups.yahoo.com/groups/orga-comp/Material/`).

Referencias

[1] Nikos Drakos, “Manual de Latex,” 1994, Computer Based Learning Unit, University of Leeds, http://www.fceia.unr.edu.ar/lcc/cdrom/Instalaciones/LaTex/latex.html.

[2] J. L. Hennessy and D. A. Patterson, “Computer Architecture. A Quantitative Approach,” 3ra Edición, Morgan Kaufmann Publishers, 2000.

[3] Curso Práctica Organización de Computadoras, “Instructivo GXEmul y NetBSD,” Univ. de Buenos Aires, 2010, .

[4] Free Software Foundation, “Core GNU utilities manual,” 2011, http://www.gnu.org/software/libc/manual/html_node/Getopt.html.

[5] MIPS Technologies, Inc., ”MIPS32 Architecture For Programmers Volume I: Introduction to the MIPS32 Architecture”, 2001, http://www.ece.lsu.edu/ee4720/mips32v1.pdf

[6] Linux, ”Linux Cross Reference”, http://lxr.free-electrons.com/source/include/asm-mips/fpregdef.h?v=2.6.25

[7] Charles Price, ”MIPS IV Instruction Set”, 1995,http://math-atlas.sourceforge.net/devel/assembly/mips-iv.pdf