Versión en C del comando rev, con lógica en assembly

Lucas Simonelli, *Padrón Nro. 93111*
lucasp.simonelli@gmail.com

Tomás Boccardo, *Padrón Nro. 93637*
tomasboccardo@gmail.com

Andrés Sanabria, *Padrón Nro. 93403*
andresg.sanabria@gmail.com

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

Resumen

El presente trabajo consiste en la implementación de un programa, similar al comando rev de Unix, encargado de concatenar y escribir en stdout el contenido invertido de cada línea, de uno o más archivos pasados como parámetros. El manejo de parámetros se realizará en C, mientras que se implementará en assembly MIPS la función reverse(fd1, fd2), que dados dos fd, realiza la inversión del primero línea a línea y lo imprime al segundo.

1. Introducción

El objetivo del trabajo desarrollado en este informe es familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, realizando en assembly MIPS una función que invierta línea a línea un archivo.

2. Función

El ejecutable tendrá el mismo objeto que el comando rev, es decir, leerá un archivo por alguno de los canales ofrecidos y lo invertirá línea a línea.

3. Desarrollo

3.1. Diseño e Implementación

Algunas suposiciones realizadas y detalles de implementación fueron los siguientes:

- Cada caracter mide 1 byte.
- El programa recibe como parámetros los archivos sobre los que trabajar. En caso de no recibir ningún parámetro, opera sobre los datos provenientes de stdin.
- Dividimos nuestro programa en 3 módulos básicos, cada uno de los cuales tiene una tarea específica.

  leerLinea (ASM): Se encarga, como su nombre lo indica, de leer caracter por caracter el contenido de una línea hasta encontrar el fin de archivo o el caracter de fin de línea. Recibe como parámetro el descriptor del archivo del que debe leer o el descriptor de stdin si este fuera el caso. Además recibe un puntero a un int donde almacenará el tamaño de la línea leida y un puntero a char pointer para almacenar los caracteres leidos. Devuelve 0 si la lectura fue exitosa o un número mayor a 0 si hubo un error.

  invertirLinea (ASM): Este módulo se utiliza para invertir el orden de aparición de los caracteres en una línea. De este modo, el último caracter de la línea quedará en primer lugar y el primero último, y de forma análoga se intercambiaran el resto de los caracteres. Recibe como parámetro un buffer con la línea en su estado original y devuelve en ese mismo buffer, la línea invertida.

  main (C): Por último, tenemos la función principal que se encarga de interpretar los parámetros con los que fue llamado el programa y llamar a los módulos antes mencionados cuando sea necesario. De acuerdo a los parámetros que le sean ingresados el programa se comportará de distinta manera:

  Si recibe como párametro '-h' muestra por pantalla una breve explicación del uso del programa.

  En caso de recibir el parámetro '-V' muestra la versión del programa en ejecución.

  De no recibir parámetros lee por entrada standard e invierte el contenido de cada línea.

  En otro caso, abre e invierte las líneas de los archivos pasados como parámetro. En caso de fallar alguno de ellos, informa por stdout el nombre del archivo que no encontró. Retorna 0 si la ejecución fue exitosa o un número mayor a 0 en otro caso.

4. Comandos para la compilacion

Para compilar el programa, deberá introducirse el siguiente comando:

```
#compilamos
~$make

#corremos
~$./tp1 [archivo] [opciones]
```

4.1. Sintaxis de uso

```
$ ./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
echo "Hola mundo" | ./tp1
```

5. Corridas de prueba

(Los archivos se facilitan junto con este informe en la entrega digital)

5.1. Corrida con archivo de parámetro

```
~$./tp1 ejemplo.txt
1elif ed aenil aremirp al se atsE
.adnuges al se atse y
```

5.2. Corrida con entrada de stdin

```
~$echo "Hola mundo" | ./tp1
odnum aloH
```

6. Código fuente

6.1. 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"


//Lee una linea de tamanio arbitrario. Devuelve NULL al llegar a EOF


void invertirLinea(char* linea,int len){
    if (!linea)
                return;
    int i = 0;
    int l = len-1;
    while (l > i){
                //Swap
                char aux = linea[i];
                linea[i]=linea[l];
                linea[l]=aux;
                i++;
                l--;
    }

}


int main(int argc, char** argv){
        //FILE* ejemplo = fopen("ejemplo","r");
        int nFiles = argc - 1;
```

```
        int file;
        bool noFile = false;


        if (nFiles == 0){
                file = 0; // stdin
                nFiles = 1;
                noFile = true;
        }
        else if (strcmp(argv[1],"-h")==0 && (nFiles==1)){
                printf("Usage:\ntp0 -h\ntp0 -V\ntp0 [file ...]\nOptions:\n-V, --
                    version   Print version and quit.\n-h, --help   Print this
                    information and quit.\nExamples:\ntp0 foo.txt bar.txt\ntp0 gz.
                    txt\n");
                return 0;
                }
        else if (strcmp(argv[1],"-V")==0 && (nFiles==1)){
                printf("Tp0 Version 1.0");
                return 0;
                }

        int i = 0;
        int res = 0;
        while (i < nFiles){
                if (! noFile){
                        file = open(argv[i+1],O_RDONLY);
                }

                if(file < 0){ // file < 0 es error
                        fprintf(stderr,"An error has occurred while opening file %
                            s\n. The program will exit now.",argv[i+1]);
                        exit(1);
                } else {
                        res=reverse(file, 1); // 1 es stdout

                        /*
                        //      LEER LINEA DEBUG MODE: ON
                        int lineLength;
                        printf("Linea: %s\n",leerLinea(file,&lineLength));
                        printf("length: %d\n",lineLength);
                        //      LEER LINEA DEBUG MODE: OFF
                        */

                        close(file);
                        if(res != 0){
                                printf(*(reverse_errmsg[res]));
                                return res;
                        }
                }
                i++;
        }
        printf(*(reverse_errmsg[res]));
        return 0;
}
```

6.2. reverse.S

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


#define STACK_SIZE 40
```

```
#define TAM_INI_CADENA 40
#define SYS_WRITE 15

.text
.align 2
.globl reverse #a0->fdin, a1->fdout
.ent reverse
reverse:        #variables locales: t0->int size
        subu sp, sp, STACK_SIZE #ra, $fp, gp + a0 a1 a2 a3+ size +1 padding
        sw $fp, STACK_SIZE-8(sp)
        sw gp, STACK_SIZE-4(sp)
        sw ra, STACK_SIZE(sp)
        sw a0, STACK_SIZE+4(sp)
        sw a1, STACK_SIZE+8(sp)

mientras:
        lw a0, STACK_SIZE+4(sp)
        li t0, t0, 0;                   # inicializo size en 0
        sw t0, STACK_SIZE-16(sp)        # guardo size en el stack
        addiu a1, sp, STACK_SIZE-16     # cargo en a1 la direccion de size en a0
            tengo el fd
        jal leerLinea                   # llamo a leer linea
        lw t0, STACK_SIZE-16(sp)        # cargo size en t0
        bnez v1, manejoError            # capturo errores en leerLinea
        bez t0, salidaExitosa           # si size<=0 entonces termino loop(eof)
        mv a0, v0                       # cargo en a0 el puntero a la linea leida
        addiu a1, sp, STACK_SIZE-16     # cargo en a1 la direccion de size
        jal invertirLinea               # llamo a invertir linea

        #Imprimo linea en fdout
        mv a1,a0                        # copio en a1 el puntero a la linea leida
        li v0, SYS_WRITE                # cargo en v0 el codigo de syscall Write
        lw a0, STACK_SIZE+8(sp)         # cargo en a0 fdout
        lw STACK_SIZE-16(sp)            # cargo en a2 el size de la linea
        syscall                         # escribo linea en fdout
        blez v0, error_escritura        # manejo error_escritura

        move a0,a1                      # cargo en a0 linea leida
        jal myfree                      # libero linea
        bnez v0, error_free2            # chequeo error en liberacion

        j mientras                      # repito mientras

error_free2:
        # ERROR_FREE
        li v1, 3                # codigo de error en 3
        b manejoError

error_escritura:
        # ERROR_ESCRITURA
        li v1, 4                # codigo de error en 4

manejoError:
        # TODO: Manejar error en v1 tengo codigo de error
        move v0,v1              # Devuelvo en v0 el codigo de error
        b popStackInv

salidaExitosa:
        li v0,0                 # Cargo codigo de exito

popStackInv:
        lw $fp, STACK_SIZE-8(sp)
```

```
        lw gp, STACK_SIZE-4(sp)
        lw ra, STACK_SIZE(sp)
        lw a0, STACK_SIZE+4(sp)
        lw a1, STACK_SIZE+8(sp)
        addiu sp, sp, STACK_SIZE
        jr ra

.end reverse

.ent invertirLinea
invertirLinea: #t0->linea, t1->len
        subu sp, sp, 8  # $fp, gp
        sw $fp, 0(sp)
        sw gp, 4(sp)
        sw a0, 12(sp)
        sw a1, 16(sp)
        move t0, a0
        move t1, a1
        beqz t0, popStackInv
        li t2, 0 #i->t2
        addiu t1, t1, -1 #t1->l-1
while:
        bgt t1, t2, popStackInv
        addu t5, t0, t2 # t5 = t0+t2
        lb t3, 0(t5) #t3 = aux = linea[i];
        addu t6, t0, t1 # t6 = t0+t1
        lb t4, 0(t6) #t4 = linea[l];
        sb t4, 0(t5) #linea[i]=linea[l];
        sb t3, 0(t6)
        addiu t2, t2, 1 #i++
        addiu t1, t1, -1 #l--
        b while

popStackInv:
        lw $fp, 0(sp)
        lw gp, 4(sp)
        lw a0, 12(sp)
        lw a1, 16(sp)
        addiu sp, sp, 8 # $fp, gp
        jr ra
.end invertirLinea


.globl leerLinea
.ent leerLinea
leerLinea:                      # a0: fd al archivo, a1: puntero a largo
        subu sp, sp, STACK_SIZE #ra, $fp, gp + a0 a1 + 3 padding
        sw $fp, STACK_SIZE-28(sp)
        sw gp, STACK_SIZE-24(sp)
        sw ra, STACK_SIZE-20(sp)
        sw s0, STACK_SIZE-16(sp)
        sw s1, STACK_SIZE-12(sp)
        sw s2, STACK_SIZE-8(sp)
        sw s3, STACK_SIZE-4(sp)
        sw s4, STACK_SIZE(sp)
        sw a0, STACK_SIZE+4(sp)
        sw a1, STACK_SIZE+8(sp)

        move s0,a0                      # fd
        move s1,a1                      # ptr a largo
        move s2,zero                    # i
```

```
        # Reservo memoria para la linea
        li a0,TAM_INI_CADENA    # tamano para reservar memoria. Parametro de
            mymalloc
        move t7,a0
        jal mymalloc                    # llamo a mymalloc
        move s3,v0                      # s3 <- return de mymalloc
        beqz s3,error_malloc    # Si v0 == NULL -> error

        # Reservo memoria para la letra
        li a0,1                         # tamano para reservar memoria. Parametro
            de mymalloc
        jal mymalloc                    # llamo a mymalloc
        move s4,v0                      # s4 <- return de mymalloc
        beqz s4,error_malloc    # Si v0 == NULL -> error

loop:
        # Leo un caracter del archivo
        li v0,SYS_read                  # Indico que la syscall es read
        move a0,s0                      # 1er param -> fd
        move a1,s4                      # 2do param -> ptr al buffer(letra
            )
        li a2,1                         # 3er param -> chars a leer (1 en
            este caso, solo leo una letra)
        syscall
        bltz v0,error_lectura

        # Copio la letra leida en la linea
        lb t1,0(s4)                     # *letra -> t1
        addu t2,s3,s2                   # linea[i] -> t2
        sb t1,0(t2)                     # linea[i] = *letra
        addi s2,1                       # i++


        # Chequeo si el indice de linea se fue de rango
        li t2,TAM_INI_CADENA
        bne t2,s2,non_realloc   # i != TAM
        move a0,s3
        sll a1,t7,2
        move t7,a1
        jal myrealloc
        beqz v0,error_malloc
        move s3,v0

non_realloc:
        li t2,10
        lw t1,0(s4)
        beq t1,t2,end_loop      # letra == '\n'
        beqz t1,end_loop                # letra == EOF
        b loop

end_loop:
        addi s2,-1                      # i-1 para pisar el ultimo char
            almacenado (\n o eof)
        addu t2,s3,s2           # linea[i] -> t2
        li t1,0
        sb t1,0(t2)                     # linea[i] = '\0'

        # Libero letra
        move a0,s4
        jal myfree
        bnez v0,error_free
```

```
        bnez s2,return_ok
        move a0,s3                      # Largo 0, libero linea
        jal myfree
        bnez v0,error_free
        b zero_length_no_error

error_lectura:
        # ERROR_LECTURA
        li v1,1         # codigo de error en 1
        b zero_length

error_malloc:
        # ERROR_MALLOC
        li v1,2         # codigo de error en 2
        b zero_length

error_free:
        # ERROR_FREE
        li v1,3         # codigo de error en 3
        b zero_length

zero_length_no_error:
        li v1,0         # codigo de error en 0
zero_length:
        move v0,zero
        sw zero,0(s1)   # Pongo el largo en el parametro
        b return

return_ok:
        move v0, s3     # Devuelvo el puntero a linea
        li v1,0         # codigo de error en 0
        sw s2,0(s1)     # Pongo el largo en el parametro
        b return

return:
        lw $fp, STACK_SIZE-28(sp)
        lw gp, STACK_SIZE-24(sp)
        lw ra, STACK_SIZE-20(sp)
        lw s0, STACK_SIZE-16(sp)
        lw s1, STACK_SIZE-12(sp)
        lw s2, STACK_SIZE-8(sp)
        lw s3, STACK_SIZE-4(sp)
        lw s4, STACK_SIZE(sp)
        lw a0, STACK_SIZE+4(sp)
        lw a1, STACK_SIZE+8(sp)
        addiu sp, sp, STACK_SIZE
        jr ra
.end leerLinea

.data
.align 2
reverse_errmsg: .word sinError, error1, error2, error3, error4
                .size reverse_errmsg, 16
                .align 0
sinError: .asciiz "error4"
error1: .asciiz "error1"
error2: .asciiz "error2"
error3: .asciiz "error3"
error4: .asciiz "error4"
```

7. Conclusiones

- Se utilizó la ABI y se aprendieron las convenciones.
- Se aprendió a compilar un programa en assembly y a linkearlo con código C.
- Uno de los puntos más importantes es la cantidad de líneas de código que se escribieron en las funciones de assembly comparadas con las de C (mucho mayor).

8. Enunciado

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/`).