Univesidad de Buenos Aires - FIUBA
66:20 Organización de Computadoras
Trabajo práctico 1: Programación MIPS

Barrera Oro, Rafael (83240)
Bacigaluppo, Ivan (98064)
Irrazabal, Diego (98125)

26 de Septiembre del 2017

Índice

1. Documentación

1.1. Diseño

Este segundo tp es más complejo que el anterior puesto que el objetivo es familiarizarse con las instrucciones de MIPS y el concepto de ABI.

Comenzamos realizando un archivo makefile bastante simple, con algunos flags, para poder compilarlo y ejecutarlo en cuanto antes. Para así poder empezar a probar cualquier cambio que hicieramos, y asegurarnos que compilara adecuadamente con el comando make. Este makefile compilaría el código mediante los siguientes comandos:

gcc -g -Wall -o tp1 main.c

Luego comenzamos con el código ya propio de lo pedido en la consigna. En primer lugar, al igual que para el TP0, armamos el main de manera que distinguiese que opción se había ingresado para ejecutar el programa. Al verificar esto, armamos una función para el comando -h, la cual imprime lo pedido; con el comando -V directamente la versión y, por último, hay una función encargada de verificar cual es el archivo de entrada, cual es el de salida del programa y cuales los buffer de entrada y salida. En los primeros dos casos el programa termina inmediatamente después de la impresión.

Luego de esto, se llama a una función handle que recibe los argumentos correspondientes como parámetros si los hubiese, o NULL en caso contrario. Esta función se encarga de procesar los archivos de entrada y salida, o asiganar los inputo o output standard para trabajar si alguno hubiera sido pasado como NULL.

Siguiendo el modelo de implementación incremental, en una primera básica versión esta función requeriría un archivo input existente, e imprimiría el resultado por la consola. De esta forma se simplifica la verificación del correcto funcionamiento del programa.

Finalmente, handle realiza la verificacion del archivo de entrada. Si lo recibido para este mismo es distinto de NULL, lo abre para lectura; en caso contrario, toma como entrada por defecto la terminal (stdin). Luego, de manera similar, verifica el archivo de salida. Si es distinto de NULL, lo abre pero para escritura en este caso y, de manera contraria, si no se le pasó por comando de entrada, utiliza por defecto la terminal como salida (stdout). Cualquier error en la apertura de los archivos se informa mediante la salida estándar stderr.

A continuación, se llama a la función is_pal, la cual está codificada en MIPS, tal como pide la consigna. Internamente esta función hace un llamado a getch, una función que se encunetra en otro módulo, en donde se procesa el archivo. Para esto se crea un buffer, utilizando la función mymalloc proporcionada por la cátedra. Luego este buffer se llena mediante una llamada a la syscall read.

Una vez que se tiene el buffer ya cargado, se pasa a leer una por una las palabras del mismo. Para esto nos encontramos con una dificultad que todavía no pudimos resolver, la cual es como separar las palabras en MIPS. Luego, cada una de las palabras debería pasar por la función strlen, la cual devuelve la longitud de ellas. Una vez que se obtiene esta longitud se pasa a un loop en el cual se verifica si la palabra es o no un palíndromo. En caso de que lo sea, se debería hacer una llamada a putch, la cual todavía no se encuentra implementada, pero que internamente debería llenar un buffer de salida y llamar a la syscall write.

Una vez que esté finalizado el código por completo procederemos a probarlo

de la misma manera que en el tp0, mediante pruebas automatizadas que se pueden correr directamente desde el makefile.

1.2. Implementación

1.2.1. void print_version()

Imprime la version del programa por consola.

1.2.2. void print_usage()

Imprime la leyenda de ayuda del programa por consola.

1.2.3. int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes)

Recibe un input file descriptor, un tamaño de bytes para los buffer de lectura, un output file descriptor y un tamañao de bytes para los buffers de salida. Internamente llama a la función implementada en MIPS is_pal.

1.2.4. void handle(char* input_file_name, char* output_file_name, int input_buffer, int output_buffer,)

Recibe punteros a los nombres completos de los archivos de entrada y salida, procede a ejecutar la logica completa del programa, mediante el llamado a la función de MIPS (si algun nombre es equivalente a "se reemplaza por la entrada/salida estandar). A su vez, recibe los tamaños de los buffer de entrada y salida.

1.2.5. int main(int argc, char** argv)

Punto de entrada al programa, se procesan los parametros recibidos de linea de comando y se ejecuta la logica del programa de ser correctos, de lo contrario se muestra la leyenda de ayuda.

1.3. Compilación

Se ha incluido un archivo Makefile para simplificar la obtención del ejecutable, el mismo puede obtenerse simplemente mediante la ejecución del comando *make*, que generará un archivo binario *tp1*:

```
$ make
gcc -g -Wall -o tp1 tp1.c
$ ls
tp1.c Makefile tp1
```

1.4. Ejecución

Una vez obtenido el ejecutable, el mismo se puede ejecutar con el parámetro *-h* para obtener la leyenda de ayuda:

```
$ ./tp0 -h
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
```

O utilizando cualquiera de los parámetros requeridos por el enunciado:

```
$ echo "somos todos bob" | ./tp1 -o pal.txt
$ cat pal.txt
somos
bob
```

2. Casos de prueba

2.1. Makefile

Se puede utilizar el Makefile para correr casos de prueba:

```
$ make test
gcc -g -Wall -o tp1 tp1.c
./tp1 -i tests/test1.in > tests/test1.res
diff tests/test1.out tests/test1.res
./tp1 -i tests/test2.in > tests/test2.res
diff tests/test2.out tests/test2.res
./tp1 -i tests/test3.in > tests/test3.res
diff tests/test3.out tests/test3.res
./tp1 -i tests/test4.in > tests/test4.res
diff tests/test4.out tests/test4.res
```

2.2. Validación de parámetros

2.2.1. Input

```
$ ./tp1 -i /tmp/noexiste.txt
No se pudo abrir el archivo de entrada: /tmp/noexiste.txt
```

2.2.2. Output

```
$ echo "bob" |./tp1 -o /tmp/
No se pudo abrir el archivo de salida: /tmp/
```

2.3. Utilizado entrada y salida standard

```
$ echo "somos bob hope"|./tp1
somos
bob
```

2.4. Utilizado archivos

```
$ echo "somos bob hope" >> test.txt
$ ./tp1 -i test.txt -o pal.txt
$ cat pal.txt
somos
bob
```

3. Código fuente

3.1. C

```
1  #define _POSIX_C_SOURCE 1
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <ctype.h>
7  #include <errno.h>
8  #include "mymalloc.h"
9  #include "is_pal.h"
10
11 #define SUCCESS 0
12 #define ERROR_INPUT_FILE 1
13 #define ERROR_OUTPUT_FILE 2
14
15 #define VERSION "2"
16
17 #define INPUT_BUFFER_DEFAULT 1
18 #define OUTPUT_BUFFER_DEFAULT 1
19 #define INPUT_FILE_DEFAULT NULL
20 #define OUTPUT_FILE_DEFAULT NULL
21
22
23 #define SUCCESS 0
24
25 void print_version() {
26         printf("%s\n", VERSION);
27         exit(SUCCESS);
28 }
29
30 void print_usage() {
31         printf("Usage:\n");
32         printf("\ttp1 -h\n");
33         printf("\ttp1 -V\n");
34         printf("\ttp1 [options]\n");
35         printf("Options:\n");
36         printf("\t-V, --version\tPrint version and quit.\n");
37         printf("\t-h, --help\tPrint this information.\n");
38         printf("\t-i, --input\tLocation of the input file.\n");
39         printf("\t-o, --output\tLocation of the output file.\n");
40         printf("\t-I, --ibuf-bytes\tByte-count of the input buffer\
               n");
41         printf("\t-O, --obuf-bytes\tByte-count of the output buffer
               \n");
42
43         printf("Examples:\n");
44         printf("\ttp1 -i ~/input -o ~/output\n");
45         exit(SUCCESS);
46 }
47
48 void handle(char* input_file_name, char* output_file_name, int
       input_buffer, int output_buffer) {
49
50         FILE* in_f;
51         FILE* out_f;
52
53         if (input_file_name != NULL){
54                 in_f = fopen( input_file_name, "r" );
55                 if (in_f == NULL) {
```

```
56                          fprintf(stderr, "ERROR:\nNo se pudo abrir
                                el archivo de entrada: %s\n",
                                input_file_name);
57                          perror("Saliendo con error"); //perror
                                imprime el mensaje de error
                                correspondiente al ERRNO
58                          exit(ERROR_INPUT_FILE);
59                  }
60          }else
61                  in_f = stdin;
62
63          if (output_file_name != NULL){
64                  out_f = fopen( output_file_name, "w" );
65                  if (out_f == NULL) {
66                          fprintf(stderr, "No se pudo abrir el
                                archivo de salida: %s\n",
                                output_file_name);
67                          perror("Saliendo con error");
68                          exit(ERROR_OUTPUT_FILE);
69                  }
70          } else
71                  out_f = stdout;
72
73          palindrome(fileno(in_f), input_buffer, fileno(out_f),
                output_buffer);
74
75          if(in_f != NULL && in_f != stdin) {
76                  fclose(in_f);
77          }
78
79          if(out_f != NULL && out_f != stdout) {
80                  fclose(out_f);
81          }
82
83          exit(SUCCESS);
84  }
85
86  int main(int argc, char** argv) {
87
88          char* ifile = INPUT_FILE_DEFAULT;
89          char* ofile = OUTPUT_FILE_DEFAULT;
90          int ibuffer = INPUT_BUFFER_DEFAULT;
91          int obuffer = OUTPUT_BUFFER_DEFAULT;
92
93          int i;
94      for (i = 1; i < argc; i++){
95                  if(strcmp(argv[i], "-V") == 0 || strcmp(argv[i], "
                        --version")==0){
96                          print_version();
97                          exit(SUCCESS);
98                  }
99          if (strcmp(argv[i], "-h")==0 || strcmp(argv[i], "--help")
                ==0){
100                 print_usage();
101                         exit(SUCCESS);
102         }
103                 if(strcmp(argv[i], "-i") == 0 && strcmp(argv[i], "
                        --input") == 0){
104                         if (strcmp(argv[i+1],"-")!=0)
105                                 ifile = argv[i+1];
106                 }
```

```
107                     if(strcmp(argv[i], "-o") == 0 && strcmp(argv[i], "
                            --output") == 0){
108                             if (strcmp(argv[i+1],"-")!=0)
109                                     ofile = argv[i+1];
110                     }
111                     if(strcmp(argv[i], "-I") == 0 && strcmp(argv[i], "
                            --ibuf-bytes") == 0){
112                             ibuffer = atoi(argv[i+1]);
113                     }
114                     if(strcmp(argv[i], "-O") == 0 && strcmp(argv[i], "
                            --obuf-bytes") == 0){
115                             obuffer = atoi(argv[i+1]);
116                     }
117             }
118             handle(ifile, ofile, ibuffer, obuffer);
119                 return SUCCESS;
120     }
```

tp1.c

3.2. Assembly (MIPS)

```
$ make asm
gcc -g -Wall -O0 -S tp1.c
Makefile tp0.c    tp1.s
```

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

    .text
    .abicalls
    .allign 2
    .global palindrome
    .ent palindrome
    .extern getch
    .extern putch

palindrome:
    .fram $fp, 40, ra
    .set noreorder
    .cpload t9
    .set reorder

    subu    sp,     sp,     52              #guardo los registros del SRA
    .cprestore 28
    sw      $fp,    24(sp)
    sw      ra,     32(sp)
    move    $fp,    sp

    sw      a0,  40($fp)                    #save arg1 ifd
    sw      a1,  44($fp)                    #save arg2 ibytes
    sw      a2,  48($fp)                    #save arg3 ofd
    sw      a3,  52($fp)                    #save arg4 obytes

    jal     getch

    move    t0,     v0                      #guardo el retorno de getch: puntero al buffer?
    move    a0,     t0                      #el puntero a la linea como argumento.
                                            #se podria hacer todo de una? is that safe?

    #si el sys_read nos tira por linea, y leemos de a bytes, deberiamos
    #cargar bytes hasta completar la palabra. Revisando contra los 'separadores'? (tipo ver al final)

    #CREO QUE lee por tam de buffer, asi que va a haber que buscar la palabra:
    #recorrer byte a byte hasta encontrar un separador,
    #siguiendo un indice SIMILAR a strlen, mas hay que controlar contra separadores

    #comienza algoritmo
    #recibe el puntero a la palabra, por a0

```

```
46      jal     strlen
47      move    t0,     v0                      #t0 = strlen()
48      subi    t1,     t0,     1               #t1: j = t0-1
49      div     t3,     t0,     2               #t3 = t0 / 2
50
51      li      t2,     1                       #t2:contador i = 1
52      li      v0,     1                       #v0 return value = 1
53  palindrome_for:
54      bge     t2,     t3      palindrome_exit #if i==len/2: exit (
            con v0=1)
55      lb      t4,     0(a0)                   #word[i] -> t4
56
57      sub     t5,     t0,     t2              #t5 = t0-t2
58      add     t6,     t5,     t1              #get indice del
            final.
59      lb      t7,     0(t6)                   #word[j] -> t7
60
61      beq     t4,     t7,     palindrome_continue
62      li      v0,     0                       #if t4 != t7: return
             value=0
63      j       palindrom_break                 #break
64
65  palindrome_continue:
66      addi    a0,     a0,     1               #siguiente indice
67      addi    t2,     t2,     1               #i++
68      j       palindrom_for
69
70  palindrome_exit:
71      #salida correcta del ciclo con v0=1 => es palindromo
72
73      #imprimir en el buffer de salida
74      #   a2: arg3 ofd
75      #   a3: arg4 obytes
76      #   hay que pasarle el puntero a la palabra
77      #   y cantidad de bytes?
78
79      move a0, a2
80      move a1, a3
81
82      jal putch
83      #ESTO IMPRIMIO UNA PALABRA? VARIAS?
84
85      #hay que hacer que imprima la palabra si es palindromo
86      #y volver a loop de buscar palabras
87
88      #y cuando el buffer esta lleno? putch llena el buffer?
            sys_write escribe directo?
89      #escribe del buffer? hay que llenar el buffer?
90
91  palindrome_break:
92      #salida por break v0=0: no es palindromo.
93      #retornar al loop de buscar palabras de una
94
95  return_to_main:
96      lw      ra,     32(sp)
97      lw      $fp,    24(sp)
98      lw      gp,     28(sp)
99      addu    sp,     sp,     40
100     jr      ra
101
102 strlen:
103     li      v0,     0                       #return value len=0
```

```
104 strlen_loop:
105     lb      t0,     0(a0)                   #load byte desde a0.
            Puntero
106     beq     t0,     0,      strlen_exit     #if 0: fin de str =>
            exit
107     addi    a0,     a0,     1               #else: mueve indice
            al siguiente
108     addi    v0,     v0,     1               #len++
109     j       strlen_loop
110
111 strlen_exit:
112     jr      ra                               #return
113
114 .data
115     separador: .asciiz " " #algo tipo asi?
116     sep2: .asciiz "$"
```

is_pal.S

4. Conclusiones

Dado que el trabajo no está terminado no hay muchas conclusiones para obtener. Lo que pudimos observar es que entender el lenguaje MIPS no es simple y lleva tiempo y trabajo. Esto se puede ver en que algo que fue muy simple en código C, como obtener palíndromos de un archivo, fue totalmente distinto al pasar a código fuente, en donde nos encontramos con muchas dificultades. A parte de esto, pudimos seguir familiarizandonos con herramientas como LaTex y la máquina virtual de gxemul, la cual nos permite levantar la imagen de NetBSD, sin la cual no podríamos trabajar con código MIPS.