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 mymalloc.S is_pal.S getch.S getwordlen.S
    putch.S palindrome.S strcopy.S clear_buf.S
$ 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:

```
$ ./tp1 -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 mymalloc.S is_pal.S getch.S getwordlen.S
    putch.S palindrome.S strcopy.S clear_buf.S
```

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

    .text
    .abicalls
    .global palindrome
    .ent palindrome

palindrome:
    .frame $fp, 48, ra
    .set noreorder
    .cpload t9
    .set reorder

    subu    sp,     sp,     64              #guardo 16 bytes sra (gp, fp, ra, padding), 16 bytes aba, 16 bytes reg s
    .cprestore 60
    sw      $fp,    56(sp)
    sw      ra,     52(sp)
    move    $fp,    sp

    move    s0,     a0                      #fd input file
    move    s1,     a1                      #guardo input buffer tam
    move    s2,     a2                      #fd output file
    move    s3,     a3                      #output buffer tam

    move    a0,     s1
    jal     save_regs                       #guardo los regs s
    jal     mymalloc                        #creo buffer de entrada
    jal     restore_regs                    #restauro regs s
    move    s4,     v0                      #guardo puntero al buffer de entrada en s4

    move    a0,     s3
    jal     save_regs                       #guardo los regs s
    jal     mymalloc                        #creo buffer de salida
    jal     restore_regs                    #restauro regs s
    move    s5,     v0                      #guardo en s5 buffer de salida
    sw      s5,     64($fp)                 #guardo el puntero al principio del buffer

    li      a0,     4096                    #buffer interno grande
    jal     save_regs                       #guardo los regs s
    jal     mymalloc                        #pide la memoria de ese buffer
    jal     restore_regs                    #restauro regs s
    move    s6,     v0                      #guarda puntero al buffer interno en s6

    li      s7,     0                       #en s7 mantengo cuantos bytes ya guarde en buf de salida

```

```
47 get_words:                                  #loop para obtener
       palabras del archivo
48     move    a0,     s0                      #input file primer
           parametro del getch
49     move    a1,     s1                      #tam buffer segundo
           parametro del getch
50     move    a2,     s4                      #puntero al buffer
           tercer parametro de getch
51     jal     save_regs                       #guardo los regs s
52     jal     getch
53     jal     restore_regs                    #restauro regs s
54     beqz    v0,     end
55     addu    s0,     v0,     s0              #adelanto el puntero a
           fd
56
57 get_next_len:                                #loop para obtener len
       de la siguiente palabra
58     lb      t0,     0(s4)
59     beqz    t0,     get_words               #se llego al final del
           buffer de entrada, hay que cargarlo nuevamente TODO: VER SI
            ES BLTZ
60
61     move    a0,     s4                      #vamos a obtener tam de
            la proxima palabra
62     jal     save_regs                       #guardo los regs s
63     jal     getwordlen
64     jal     restore_regs                    #restauro regs s
65
66     move    a0,     v0                      #guardo resultado como
           primer param de strcopy
67     move    a1,     s4                      #buffer a copiar,
           segundo parametro
68     move    a2,     s6                      #buffer donde debe
           copiar, tercer parametro
69     jal     save_regs                       #guardo los regs s
70     jal     strcopy
71     jal     restore_regs                    #restauro regs s
72     addu    s4,     v0,     s4              #adelanto buffer de
           entrada
73     addi    s4,     s4,     1
74
75     move    a0,     s6                      #llamada a is_pal, con
           el buffer interno
76     jal     save_regs                       #guardo los regs s
77     jal     is_pal
78     jal     restore_regs                    #restauro regs s
79     beqz    v0,     get_next_len            #si no es pal retorno
           al main
80
81 save_words:                                 #guardar palabras a
       imprimir en buffer de salida
82     move    a0,     s6
83     jal     save_regs                       #guardo los regs s
84     jal     getwordlen                      #obtengo len de la
           palabra a guardar en buf de salida
85     jal     restore_regs                    #restauro regs s
86
87     move    a0,     v0                      #preparo parametros
           strcopy
88     move    a1,     s6
89     move    a2,     s5
90     jal     save_regs                       #guardo los regs s
```

```
91      jal     strcopy                         #copio la palabra en el
            buffer de salida
92      jal     restore_regs                    #restauro regs s
93      addu    s5, v0, s5                      #adelanto buffer de
            salida
94
95      addu    s7, v0, s7                      #sumo bytes copiados a
            s7
96      bgt     s7, s3, print                   #si llegue a la cant de
            bytes del buf de salida imprimo
97
98  print:
99      move    a0, s2                          #fd output file primer
            parametro del putch
100     move    a1, s3                          #tam buffer a imprimir
            segundo parametro del putch
101     lw      a2, 64($fp)                     #buffer a imprimir
            tercer parametro del putch
102     jal     save_regs                       #guardo los regs s
103     jal     putch
104     jal     restore_regs                    #restauro regs s
105
106     jal     get_next_len
107
108 end:
109     move    a0, s2                          #fd output file primer
            parametro del putch
110     move    a1, s3                          #tam buffer a imprimir
            segundo parametro del putch
111     lw      a2, 64($fp)                     #buffer a imprimir
            tercer parametro del putch
112     jal     save_regs                       #guardo los regs s
113     jal     putch
114     jal     restore_regs                    #restauro regs s
115
116     jal     return
117
118
119 save_regs:
120     sw      s0, 48(sp)
121     sw      s1, 44(sp)
122     sw      s2, 40(sp)
123     sw      s3, 36(sp)
124     sw      s4, 32(sp)
125     sw      s5, 28(sp)
126     sw      s6, 24(sp)
127     sw      s7, 20(sp)
128     jr      ra
129
130 restore_regs:
131     lw      s0, 48(sp)
132     lw      s1, 44(sp)
133     lw      s2, 40(sp)
134     lw      s3, 36(sp)
135     lw      s4, 32(sp)
136     lw      s5, 28(sp)
137     lw      s6, 24(sp)
138     lw      s7, 20(sp)
139     jr      ra
140
141 return:
142     move    a0, s4
```

```
143     jal     myfree
144
145     move    a0,     s5
146     jal     myfree
147
148     move    a0,     s6
149     jal     myfree
150
151     lw      ra,     52(sp)
152     lw      $fp,    56(sp)
153     lw      gp,     60(sp)
154     addu    sp,     sp,     64
155     jr      ra
156
157 .end palindrome
```

palindrome.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.