Trabajo práctico #1: conjunto de instrucciones MIPS

Santiago Fernandez, *Padrón Nro. 94.489*
fernandezsantid@gmail.com

Francisco Landino, *Padrón Nro. 94.475*
landinofrancisco@gmail.com

Matias Duarte, *Padrón Nro. 92.186*
duarte.mati@gmail.com

2do. Cuatrimestre de 2014
66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Este trabajo práctico trata de una versión en lenguaje C, de un programa que lee desde un archivo de texto o desde STDIN, un texto que contiene tags, posiblemente anidados, y verifica que la estructura sea correcta. Además, se hizo un version en Assembly, de una función, para familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI.

Índice

1. Introducción

Como objetivo de este trabajo practico se trabajo tanto usando el conjunto de instrucciones de MIPS32 como en el lenguaje C, para codificar un programa que recibe desde STDIN o desde un archivo de texto, un texto en el cual se encuentran tags, de la forma *<Tag>... </Tag>*, posiblemente anidados. El programa se encarga de verificar que estén correctamente relacionados, lo que significa que el texto que abre un tag sea igual al texto que lo cierra y que estén correctamente anidados.

También se trabajo usando el ABI presentado por la cátedra, para guardar los argumentos en el stack entre las llamadas de función a función.

2. Desarrollo

Para desarrollar este trabajo primero se realizo un programa en lenguaje C, el cual contempla las siguientes condiciones.

Los *tags* tienen que cumplir con la siguiente estructura:

- Todos los *tags* abiertos se cierran,
- Todos los *tags* que se cierran fueron abiertos.
- Todos los *tags* abiertos dentro de otro *tag* se cierran antes que ?este.
- Si se detecta una violación de alguna de las reglas anteriores, se especifica el error (*tag* sin cerrar, *tag* sin abrir, *tag* mal anidado) y el numero de línea (empezando por 0) por **stderr**. En caso de que la estructura del archivo sea valida, el programa debe devolver 0, caso contrario devolver 1, ademas de la salida por **stderr**.

Un *tag* esta compuesto por una secuencia de apertura, de la forma "<**nombre-tag**>", un contenido que puede ser texto u otros *tags*, y una secuencia de cierre de la forma "</**nombre-tag**>".

Se considera que no pueden haber caracteres del tipo *'<'* o *'>'* sin que implique una apertura o cierre de un *tag*.

Luego a partir de esta implementacion en C, se desarrollaron las funciones *validate* y *analizarTag* en Assembly MIPS32. Para dicha implementación se uso el ABI que enseño la cátedra durante las clases. Al principio de cada función se crea un stack, donde se guardan los registros del Return Address, Frame Pointer y Global Pointer en las posiciones mas grandes del stack. Luego se guardan los atributos de la función y finalmente los argumentos.

2.1. Recursos y Portabilidad

Uno de los objetivos del trabajo práctico es poder probar la portabilidad del programa en diferentes entornos. En el enunciado se pedía que el programa se pudiera ejecutar en NetBSD[4]/pmax (usando el simulador GXemul[5]) y en la versión de Linux (Knoppix, RedHat, Debian, Ubuntu) usada para correr el simulador, Linux/i386. En particular, se lo probó en Ubuntu 14.04. En GXemul

se corrió una máquina de arquitectura MIPS cuyo sistema operativo era una versión reciente de NetBSD/pmax. La transferencia de archivos entre la máquina host y la guest se hizo mediante *SSH*. Se procedió de la siguiente manera:

Para trabajar con el GXemul se procedió primero creando una nueva interfaz de red (debe crearse cada vez que se inicia el *host* y con permisos de administrador):

hostOS$ sudo ifconfig lo:0 172.20.0.1

Luego se ejecutó el GXemul en modo X:

hostOS$ ./xgxemul -e 3max -d netbsd-pmax.img -x

Una vez ya ingresado con el usuario y la contraseña en la máquina simulada, se creó un túnel reverso para saltear las limitaciones propias del GXemul:

guestOS$ ssh -R 2222:127.0.0.1:22 usuario@172.20.0.1

A partir de ese momento y dejando lo anterior en segundo plano, ya se puedo trabajar mediante SSH de manera más cómoda:

hostOS$ ssh -p 2222 root@127.0.0.1

2.2. Implementación

El programa en lenguaje C, cuenta con dos archivos: *main.c* y *validate.c*.

En el archivo *main.c* se encuentran tres funciones:

void cargarArchivoAMemoria(FILE archivoEntrada, char* text)*

Se encarga de pasar el archivo de entrada a memoria. Por *archivoEntrada* se pasa el puntero al archivo y por *text* se obtiene el puntero a la cadena de texto en memoria.

void printManual()

Imprime el manual sobre como usar el programa.

void checkFile(FILE file)*

Verifica si el archivo del cual se quiere levantar es correcto.

Luego en el main el programa se encarga de procesar todas las opciones, que se le introducen por argumento, y en el caso de que se pueda levantar el texto correctamente, llama a la función *int validate(char *text, char **errmsg)* la cual se encarga de validar el archivo. Si validate devuelve 0, el programa imprime un 0 y termina su ejecución, pero si validate devuelve un 1, imprime un 1, junto

con el mensaje de error en errmsg.

En el archivo *validate.c* se encuentran dos funciones:

int analizarTag(char text, char* tagEncontrado, int pos, int *contadorLineas)*

Analiza que el tag que le pasan por argumento sea igual al próximo tag que se cierra. Si encuentra que se abre otro tag antes de encontrar que se cierra un tag, levanta el tag anidado y se llama recursivamente para que lo verifique.

text es el puntero al texto que se esta analizando.

tagEncontrado es el puntero a la cadena de texto que contiene el tag que se levanto anteriormente y que hay que comparar con el siguiente tag que se cierre.

pos es la posición a partir de la cual hay que ir analizando el texto.

contadorLineas es el puntero al contador que lleva el numero de linea.

La función devuelve:

-1 si es un error donde el ultimo tag abierto no fue cerrado.

-2 si es un error donde el tag cerrado no corresponde con el tag abierto.

Sino devuelve un numero mayor a cero, el cual indica la posición del texto donde quedo después de analizar que se cerro correctamente el tag.

*int validate(char *text, char **errmsg)*

Va recorriendo el texto hasta que encuentre el fin o hasta que encuentre un tag. Cuando encuentra un tag, lo guarda en el atributo *tagALevantar* y luego llama a la función *analizarTag* para que verifique si se cierra correctamente. Si analizar tag devuelve en un numero mayor a cero, continua analizando el texto desde esa posición, sino verifica que tipo de error es y devuelve por errmsg, el mensaje que se va a imprimir.

text es el puntero al texto a verificar.

errmsg es el puntero a la cadena de texto donde se tiene que guardar el mensaje de error en el caso de que haya alguno error en la estructura de los tags en el texto.

La función devuelve un 0 si llego al fin del texto y no se encontró ningún error, con la estructura de los tags, o devuelve un 1 si se encontró algún error.

2.3. Compilación

Para compilar el trabajo práctico, se tiene que ejecutar la siguiente línea en la terminal de Linux.

gcc -Wall -pedantic -std=c99 main.c auxiliares.c -I / -o tp0

Para simplificar este proceso, y no tener que escribir el comando entero, se creó un archivo *compile.sh*, mediante el cual, con solo escribir *./compile.sh*, se compila el trabajo práctico.

2.4. Corrida de Pruebas

Hicimos corridas con diferentes textos y los resultados fueron los siguientes.

```
Texto 1:

<Tag1> texto </Tag1>
<Tag2>  texto2  </Tag2>
<Tag3> texto3 </tag>

Resultado:

1
Linea: 2. Tag mal anidado, el ultimo tag cerrado, no
corresponde con el ultimo tag abierto.

Texto 2:

<Tag1> texto </Tag1>
<Tag2>  texto2  </Tag2>
<Tag3> <otroTag> <italic> texto3 </italic> </otroTag> </Tag3> texto4 </Tag5>

Resultado:

1
Linea: 2. El tag abierto, no fue cerrado.

Texto 3:

<Tag1> texto </Tag1>
<Tag2>  texto2  </Tag2>
<Tag3> <Bold> <UP> texto3 </UP> </Bold> </Tag3>

Resultado:

0
```

3. Diagrama de Stack

3.1. Validate

Stack Size	56
Padding	50
ra	48
fp	44
gp	40
VALIDATE RT	36
VALIDATE BF	32
VALIDATE TAGL	28
VALIDATE J	24
VALIDATE CL	20
VALIDATE I	16
padding	12
padding	8
**errmsg	4
*text	0

3.2. Analizar Tag

Stack Size	40
Padding	36
ra	32
fp	28
gp	24
VAR AUX	20
TAGALEVANTAR	16
CONTADORLINEAS	12
POSICION	8
TAGENCONTRADO	4
*texto	0

4. Conclusiones

De este trabajo practico se pudo aprender como programar con el conjunto de instrucciones Assembly de MIPS32, asi como tambien la utilizacion correcta de la ABI de la catedra, entendiendo de esta manera como funciona una computadora a bajo nivel.

5. Referencias

[1] GXemul, http://gavare.se/gxemul/.
[2] The NetBSD project, http://www.netbsd.org/.
[3] System V application binary interface, MIPS RISC processor supplement(third edition). Santa Cruz Operations, Inc.
[4] func call conv.pdf, en el area de Material de los archivos del grupo de Yahoo.