66:20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI[1], escribiendo un programa portable que resuelva el problema descripto en la sección 5.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, 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 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, usando la herramienta TEX/ LATEX.

4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

[1] Application binary interface

5. Programa

Se trata de una versión en lenguaje C de un programa que lee un archivo de texto que contiene *tags* posiblemente anidados, y verifica que la estructura sea correcta. Esto quiere decir que:

- 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 éste.
- 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 número de línea (empezando por 0) por `stderr`. En caso de que la estructura del archivo sea válida, el programa debe devolver `0`, caso contrario devolver `1`, además de la salida por `stderr`.

Un *tag* está 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>`.

5.1. Comportamiento deseado

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ validate -h
Usage:
  validate -h
  validate -V
  validate [options] file
Options:
  -h, --help     Prints usage information.
  -V, --version  Prints version information.
  -i, --input    Path to input file (-i - for stdin)
Examples:
  validate -i -
  validate myfile.tagged
  validate -i myfile.tagged
```

Ahora usaremos el programa para validar unos archivos:

```
$ cat file1.tag
<tag1> Este es el primer tag </tag1>
Acá no hay nada
<amigou> Este es el segundo tag
<tag3> Este es el tercer tag
</tag3> </amigou>
```

```
$ ./validate file1.tag
$ echo $?
0
$ cat file2.tag
<tag1> Goto considered harmful
<tag2> Texto tag2
</tag1> Mal cerrado
</tag2>

$ cat file2.tag | ./validate -i -
Error en línea 2: tag1 cerrado antes que tag2
$ echo $?
1
$
```

El programa deberá retornar un error si se le pasa un nombre de archivo inexistente.

6. Implementación

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

6.1. API

Gran parte del programa estará implementada en lenguaje C. Sin embargo, la función que hace la validación en sí estará implementada en Assembly MIPS32, para proveer soporte específico en nuestra plataforma principal de desarrollo, NetBSD/pmax.

Una manera posible de implementar esta validación es implementando una pila de `tags` en proceso, y otra es mediante una función recursiva. Se asume que los archivos se pueden cargar completos en memoria.

El programa en C deberá procesar los argumentos de entrada, subir el texto a memoria, y llamar a la función

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

que debe estar escrita en Assembly MIPS32, donde `*text` es un puntero al texto contenido en el archivo, y `**errmsg` es un puntero a un array de caracteres, a llenar por la función `validate` en caso de error. La función debe retornar 0 en caso de que la validación sea correcta, o 1 en caso de que no. Si se hacen funciones auxiliares (como en el caso de usar recursión), también deben estar escritas en Assembly MIPS32.

6.2. Portabilidad

Pese a contener fragmentos en assembler MIPS32, es necesario que la implementación desarrollada provea un grado mínimo de portabilidad.

Para satisfacer esto, el programa deberá proveer dos versiones de `validate(char *text, char **errmsg)`, incluyendo la versión MIPS32, pero también una versión C, pensada para dar soporte genérico a aquellos entornos que carezcan de una versión más específica.

6.3. ABI

El pasaje de parámetros entre el código C (`main()`, etc) y la función `validate(char *text, char **errmsg)`, en assembler, deberán hacerse usando la ABI explicada en clase [4]: los argumentos correspondientes a los registros `$a0`-`$a3` serán almacenados por el *callee*, siempre, en los 16 bytes dedicados de la sección "function call argument area" [3].

7. Proceso de Compilación

En este trabajo, el desarrollo se hará parte en C y parte en lenguaje Assembly. Los programas escritos serán compilados o ensamblados según el caso, y posteriormente enlazados, utilizando las herramientas de GNU disponibles en el sistema NetBSD utilizado. Como resultado del enlace, se genera la aplicación ejecutable.

8. Informe

El informe deberá incluir:

- Este enunciado;
- Documentación relevante al diseño e implementación del programa, incluyendo un diagrama del stack;
- Corridas de prueba, con los comentarios pertinentes;
- Diagramas del stack de las funciones;
- El código fuente completo, en formato digital[2].

[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.

9. Fecha de entrega

- Primera entrega: Jueves 9 de Octubre de 2014
- Devoluciones primera entrega: Jueves 16 de Octubre de 2014
- Vencimiento: Jueves 23 de Octubre de 2014

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 área de Material de los archivos del grupo de Yahoo.