66:20 Organización de Computadoras
Trabajo práctico #1: programación MIPS
2^o cuatrimestre de 2017

`$Date: 2017/09/19 09:29:47 $`

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 trabajo 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 5), la presentación de los resultados obtenidos explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

4. Descripción

En este trabajo, se reimplementará parcialmente en assembly MIPS el programa desarrollado en el trabajo práctico anterior.

Para esto, se requiere reescribir el programa, de forma tal que quede organizado de la siguiente forma:

- Arranque y configuración del programa: procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores. Desde aquí se invocará a las función de procesamiento del *stream* de entrada.

- Procesamiento: contendrá el código MIPS32 assembly con la función `palindrome()`, encargada de identificar, procesar e imprimir los componentes léxicos que resulten ser palíndromos, de forma equivalente a lo realizado en el TP anterior.

La función MIPS32 `palindrome()` antes mencionada se corresponderá con el siguiente prototipo en C:

```
int palindrome(int ifd, size_t ibytes, int ofd, size_t obytes);
```

Esta función es invocada por el módulo de arranque y configuración del programa, y recibe en `ifd` y `ofd` los descriptores abiertos de los archivos de entrada y salida respectivamente.

Los parámetros `ibytes` y `obytes` describen los tamaños en bytes de las unidades de transferencia de datos desde y hacia el kernel de NetBSD, y permiten implementar un esquema de *buffering* de estas operaciones de acuerdo al siguiente diagrama:

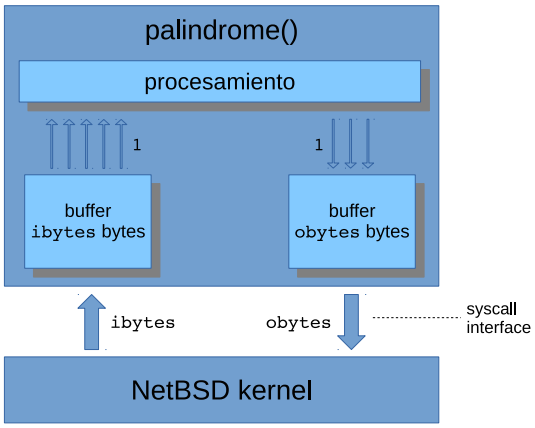

Figura 1: arquitectura de procesamiento.

Como puede verse en la figura 1, la lógica de procesamiento de la función `palindrome()` va leyendo los caracteres del *buffer* de entrada en forma individual.

En el momento en el cual `palindrome()` intente extraer un nuevo caracter, y el *buffer* de entrada se encuentre vacío, deberá ejecutar una llamada al sistema operativo para realizar una lectura en bloque y llenar completamente el buffer, siendo el tamaño de bloque igual a `ibytes` bytes.

De forma análoga, `palindrome()` irá colocando uno a uno los caracteres de las palabras capicúa en el *buffer* de salida. En el momento en el que se agote su capacidad, deberá vaciarlo mediante una operación de escritura hacia el kernel de NetBSD para continuar luego con su procesamiento.

Al finalizar la lectura y procesamiento de los datos de entrada, es probable que exista información esperando a ser enviada al sistema operativo. En ese caso `palindrome()` deberá ejecutar una última llamada al sistema con el fin de vaciar completamente el *buffer* de salida.

Se sugiere encapsular la lógica de *buffering* de entrada/salida con funciones, `getch()` y `putch()`. Asimismo durante la clase del martes 19/9 explicaremos la función `mymalloc()` que deberá ser usada para reservar dinámicamente la memoria de los buffers.

4.1. Ejemplos

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

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 [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:
  tp0 -i ~/input -o ~/output
```

Codificamos un archivo vacío (cantidad de bytes nula):

```
$ touch /tmp/zero.txt
$ tp0 -i /tmp/zero.txt -o /tmp/out.txt
$ ls -l /tmp/out.txt
-rw-r--r--  1 user group 0 2017-03-19 15:14 /tmp/out.txt
```

Leemos un *stream* cuyo único contenido es el caracter ASCII `M`,

```
$ echo Hola M | tp0
M
```

Observar que la salida del programa contiene aquellas palabras de la entrada que sean palíndromos (`M` en este caso).

Veamos que sucede al procesar archivo de mayor complejidad:

```
$ cat entrada.txt
Somos los primeros en completar el TP 0.

Ojo que La fecha de entrega del TP0 es el martes 12 de septiembre.
$ tp0 -i entrada.txt -o -
Somos
Ojo
```

4.2. Interfaz

A fin de facilitar la corrección y prueba de los TPs, normalizaremos algunas de las opciones que deberán ser provistas por el programa:

- `-i`, o `--input`, permite especificar la ubicación del archivo de entrada, siendo `stdin` o cuando el argumento es "`-`", o bien cuando no haya sido especificado explícitamente en la línea de comandos (valor por defecto).
- `-o`, o `--output`, para definir la ubicación del archivo de salida en forma análoga al punto anterior. Por defecto, el programa deberá escribir sobre `stdout`. Lo mismo sucederá cuando el argumento pasado es "`-`".
- `-I`, o `--ibuf-bytes` determina el tamaño en bytes del *buffer* de entrada. El valor por defecto a usar es `1`.
- `-O`, o `--obuf-bytes` nos permite dimensionar el *buffer* se salida. El valor por defecto también es `1`.

5. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño e implementación del programa.
- Comandos para compilar el programa.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, el cual también deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas).
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

6. Fechas

- Entrega: 26/9/2017.
- Vencimiento: 10/10/2017.