Trabajo Práctico 1
Conjunto de instrucciones MIPS

Lucas Medrano, *Padrón Nro. 99247*
lucasmedrano97@gmail.com
Federico Álvarez, *Padrón Nro. 99266*
fede.alvarez1997@gmail.com

Grupo Nro. - 2do. Cuatrimestre de 2018
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

En este trabajo se quiere desarrollar un programa escrito en lenguaje C que implementa un algoritmo de Quicksort. Dicho programa ordena alfabéticamente o numéricamente las líneas de un archivo `.txt`. Se visualizará en pantalla tanto el resultado como los errores que se produzcan. El algoritmo de Quicksort tendrá una implementación en assembler MIPS32, además de la versión en C, para la cual se empleará la convención de pasaje de parámetros establecida en la ABI explicada en clase.

Índice

1. Enunciado

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, y se valorarán aquellos escritos 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 [3]. GXemul se puede hacer correr bajo Windows, en el entorno Cygwin [2].

[1] Application binary interface

5. Programa

Se trata de una versión en lenguaje C del algoritmo de Quicksort [5]. El programa recibirá como argumento el nombre de archivo cuyos contenidos se deben ordenar, y dará por `stdout` (o escribirá en un archivo) los valores ordenados. De haber errores, los mensajes de error deberán salir exclusivamente por `stderr`. Se asume que las cadenas de caracteres a ordenar aparecen de a una por línea.

5.1. Comportamiento deseado

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

```
$ qsort -h
Usage:
  qsort -h
  qsort -V
  qsort [options] archivo
Options:
  -h, --help     Imprime ayuda.
  -V, --version  Versión del programa.
  -o, --output   Archivo de salida.
  -n, --numeric  Ordenar los datos numéricamente en vez de alfabéticamente.
Examples:
  qsort -n numeros.txt
```

Ahora usaremos el programa para ordenar un archivo. Supongamos que tenemos un archivo con el siguiente contenido:

```
$ cat numeros.txt
1
3
5
7
9
2
4
6
8
10
$
```

Invocamos al programa, usando "-" como argumento de `-o` para indicarle al programa que imprima el resultado por `stdout`:

```
$ qsort -o - numeros.txt
1
10
2
```

```
3
4
5
6
7
8
9
$
```

Si lo llamamos con la opción `-n`, la salida ordena los números:

```
$ qsort -n -o - numeros.txt
1
2
3
4
5
6
7
8
9
10
$
```

El programa deberá detectar condiciones de error y reportarlas por `stderr`.

6. Implementación

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

6.1. Portabilidad

Pese a contenter 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 `qsort()`, incluyendo la versión MIPS32, pero también una versión C, pensada para dar soporte genérico a aquellos entornos que carezcan de una versión más específica.

6.2. API

Gran parte del programa estará implementada en lenguaje C. Sin embargo, la función `qsort()` estará implementada en assembler MIPS32, para proveer soporte específico en nuestra plataforma principal de desarrollo, NetBSD/pmax.

El propósito de `qsort()` es ordenar un arreglo de cadenas de caracteres, utilizando el algoritmo `quicksort`[5]. Este arreglo está organizado como una serie de punteros consecutivos a punteros a char, siendo el primero `*izq` y el último `*der`. Si `num` vale 0, las cadenas de caracteres se deberán comparar alfabéticamente, y si vale distinto de 0 se deberá interpretarlas como enteros.

```
void qsort(char** izq, char** der, int num);
```

El programa en C deberá procesar los argumentos de entrada, invocar a `qsort`, y escribir en `stdout` o un archivo el resultado. La función `qsort()` se debe implementar preferentemente de manera recursiva.

6.3. ABI

El pasaje de parámetros entre el código C (`main()`, etc) y la rutina `qsort()`, en assembler, deberá hacerse usando la ABI explicada en clase: 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" [4].

6.4. Algoritmo

El algoritmo a implementar es Quicksort [5].

7. Proceso de Compilación

En este trabajo, el desarrollo se hará parte en C y parte en lenguaje Assembler. 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 la función;

- El código fuente completo, en dos formatos: digitalizado[2] e impreso en papel.

9. Fecha de entrega

La última fecha de entrega y presentación es el jueves 11 de Octubre de 2018.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] Installing the MIPS Environment over Cygwin on Windows, `http://faculty.cs.tamu.edu/bettati/Courses/410/2006B/Projects/gxemulcygwin.html`

[3] The NetBSD project, `http://www.netbsd.org/`.

[4] System V application binary interface, MIPS RISC processor supplement (third edition). Santa Cruz Operations, Inc.

[5] Algoritmo de Quicksort, `http://www.itl.nist.gov/div897/sqg/dads/HTML/quicksort.html`.

[2] En caso de presentar un repositorio tipo GitHub, hacer un release.

2. Desarrollo

El programa puede tomar opciones de entrada para indicar el tipo de ordenamiento (alfabético o numérico) y argumentos que designan la salida y el archivo a ordenar.

2.1. Implementación

Los errores se definen como cadenas de caracteres constantes. Los mensajes de error se llaman con una función a la que se le pasa el mensaje a mostrar y el número que le corresponde al error.

Las invocaciones a la línea de comandos (como el pedido de versión o de ayuda) también se guardan en cadenas. El tipo de mensaje a mostrar es pasado a la función que los visualiza.

El programa podrá tomar sólo un parámetro, en caso que se desee acceder a la ayuda o a la versión, y un máximo de cuatro cuando se quiera ordenar un archivo. El primero de estos será el tipo de ordenamiento, que se considerará alfabético por defecto, por lo que este parámetro puede no estar; luego vendrá el output (`-o` o `--output`); el tercero será el archivo donde se escribirá el resultado del ordenamiento (`stdout` en caso de que se ingrese `-`); y por último el archivo a ordenar. Ingresar estos parámetros en cualquier otro orden resultará en un mensaje de error.

Al inicio de la función `main` se verificarán los argumentos ingresados para ver si coinciden con las opciones requeridas para que el programa pueda responder de acuerdo con el comportamiento elegido.

Se decidió por implementar una versión *in place* del algoritmo Quicksort, por cuestiones de simplicidad y para mayor legibilidad el código. Ésta diferencia el tipo de ordenamiento que se hará en un método de ordenamiento general, lo cual se logra con un entero identificador: si vale 1, el ordenamiento es numérico; para cualquier otro valor se realizará un ordenamiento alfabético. En el primer caso se utilizará una función atoi para pasar las cadenas de caracteres a números enteros mientras que en el otro caso se compararán las cadenas entre sí.

Se elegió cambiar el nombre de la función qsort puesto que esta se encuentra reservada por C. Se llamará orgaqsort a la función implementada en C, y orgaqsortassembly a su contraparte en assembly.

El comando utilizado para compilar el programa será `gcc -Wall -o qsort qsort.c orgaqsortassembly.s`

2.1.1. Diagramas de stack

A continuación se muestran los diagramas de stack de las diferentes funciones del archivo assembly.

2.2. Pruebas

2.2.1. Entradas incorrectas

Ingresando sólo el nombre del programa ejecutable:

```
$ ./qsort
qsort: La cantidad de parametros no es la correcta.
Intente 'qsort -h' para mas informacion
```

Ingresando un argumento inválido:

```
$ ./qsort -w
qsort: La combinacion de parametros no es la correcta.
Intente 'qsort -h' para mas informacion
```

Ingresando más argumentos de los necesarios:

```
$ ./qsort f f f f f
qsort: La cantidad de parametros no es la correcta.
Intente 'qsort -h' para mas informacion
```

Ingresando un archivo que no existe:

```
$ ./qsort -o - numbers.txt
El archivo que quiere ordenar no existe
```

Ingresando el orden incorrecto de parámetros (primero `-o` y luego `-n`):

```
$ ./qsort -o -n - numeros.txt
qsort: La combinacion de parametros no es la correcta.
Intente 'qsort -h' para mas informacion
```

Aquí detecta que la opción de ordenamiento alfabético tiene argumentos de más.

2.2.2. Ordenando líneas

Ingresando un archivo para ordenar alfabéticamente:

```
$ ./qsort -o - zeta.txt
zzzzzzzzzzzz a
zzzzzzzzzzzz b
zzzzzzzzzzzz sabia que Asuntos Internos le tendia una trampa
```

```
$ ./qsort -o - text1.txt
 aaaaaaa
aaaaaaa
aaaaaaa a
```

```
aaaaaaa b
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
hola

que
tal
```

Ingresando el nombre del archivo que se desea para salida de un ordenamiento alfabético:

```
$ ./qsort -o orden_alfabetico.txt numeros.txt
```

Se genera un archivo con el nombre `orden_alfabetico.txt` que contiene las palabras de `numeros.txt` en orden alfabético.

```
$ cat orden_alfabetico.txt
1
10
2
3
4
5
6
7
8
9
```

Ingresando un archivo para ordenar numéricamente:

```
$ ./qsort -n -o - numeros.txt
1
2
3
4
5
6
7
8
9
10
```

Ingresando el nombre del archivo que se desea para salida de un ordenamiento numérico:

```
$ ./qsort -n -o orden_numerico.txt zeta.txt
```

Se genera un archivo con el nombre `orden_alfabetico.txt` que contiene las palabras de `zeta.txt` en orden numérico.

```
$ cat orden_numerico.txt
zzzzzzzzzzzz sabia que Asuntos Internos le tendia una trampa
zzzzzzzzzzzz a
zzzzzzzzzzzz b
```

Por cuestiones de espacio no se muestran los resultados de correr el programa con los archivos `alice.txt` y `quijote.txt`.

3. Conclusiones

La realización de este trabajo resultó de mucha utilidad para entender como funcionan las herramientas usadas, como el emulador y el lenguaje, algo que se presentó dificil en un principio. También contribuyó para incorporar las convenciones usadas para trabajar con a arquitectura MIPS. Por otra parte, permitió aprender a programar en lenguaje ensamblador, prestándole atención a los detalles y valiéndose de las estructuras generadas por las funciones, como la pila, para almacenar los parámetros cuyo valor se quiere seguir a lo largo de la ejecución de esa función en particular. Sumado a esto, sirvió para comprender cómo combinar archivos en C y archivos en lenguaje assembly, cosa inédita hasta ese momento. Sacamos del trabajo buenas prácticas, como lo son el probar las funciones del codigo assembly por separado para luego incorporarlas, ya que una vez que está todo junto se hace mas complejo; y el uso del debugger tanto para solucionar los problemas que pudiera tener el programa como para realizar el seguimiento del código ejecutado.

Se entregan dos scripts. Uno en C, con una implementacion de qsort en C y otras funciones necesarias; y uno en assembly sólo con la funciones `qsort`, `comparar` y `atoi`.