Trabajo Práctico 1
Conjunto de instrucciones MIPS

Lucas Medrano, *Padrón Nro. 00.000*
`dirección de e-mail`
Federico Álvarez, *Padrón Nro. 00.000*
`dirección de e-mail`

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 **1**

2. Desarrollo **6**
2.1. Implementación 6
2.2. Pruebas 6
2.2.1. Entradas incorrectas 6
2.2.2. Ordenando líneas 7
2.3. Diagramas de stack 8

3. Conclusiones **8**

4. Código fuente **9**
4.1. Código en C 9
4.2. Código Assembly 12

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 Quicksort diferencia el tipo de ordenamiento que se hará en un método de ordenamiento general. Esto 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. En el otro caso se compararán las cadenas entre sí.

Al inicio de la función `main` se verifican los argumentos ingresados para ver si coinciden con las opciones requeridas. Así se puede responder de acuerdo con el comportamiento elegido.

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
zzzzzzzzzzz a
zzzzzzzzzzz b
zzzzzzzzzzz sabia que Asuntos Internos le tendia una trampa

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

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.

Ingresando un archivo para ordenar numéricamente:

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

$ 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 numeros.txt
```

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

2.3. Diagramas de stack

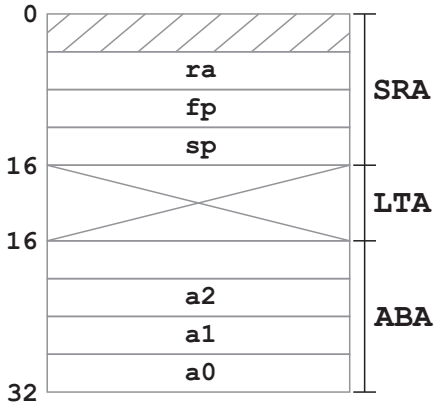

Figura 1: Diagrama de stack de `qsort`.

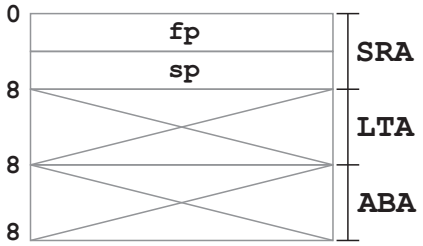

Figura 2: Diagrama de stack de `comparar`.

3. Conclusiones

4. Código fuente

4.1. Código en C

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <mips/regdef.h>

#define salida_error_parametros 1
#define salida_error_archivo_inexistente 2
#define mensaje_error_cantidad_parametros "qsort: La cantidad de parametros
    no es la correcta.\nIntente 'qsort -h' para mas informacion"
#define mensaje_error_parametros "qsort: La combinacion de parametros no es
    valida.\nIntente 'qsort -h' para mas informacion"
#define mensaje_error_archivo_inexistente "El archivo que quiere ordenar no
    existe"
#define MENSAJE_AYUDA "Usage:\n\tqsort -h\n\tqsort -V\n\tqsort [options]
    archivo\nOptions:\n\t-h, --help \tImprime ayuda.\n\t-V, --version \
    tVersion del programa.\n\t-o, --output \tArchivo de salida.\n\t-n, --
    numeric \tOrdenar los datos numericamente en vez de alfabeticamente.\
    nExamples:\n\tqsort -n numeros.txt"
#define VERSION "v1.0"


extern void orgaqsortassembly(char** izq, char** der, int num);


void mostrar_error_y_salir(char *mensaje_error, int numero_salida){
    fprintf(stderr, "%s\n", mensaje_error);
    exit(numero_salida);
}


void mostrar_version(){printf("%s\n", VERSION);}


void mostrar_ayuda(){printf("%s\n", MENSAJE_AYUDA);}


int obtener_longitud_maxima(FILE* archivo){
    //Obtiene longitud contando el \n o \0 del final
    // ejemplo: para "hola", maximo = 5
    int maximo = 0;
    int actual = 0;
    int caracter;
    while ((caracter = fgetc(archivo)) != EOF){
        actual++;
        if (caracter == '\n'){
            if (actual > maximo) maximo = actual;
            actual = 0;
        }
    }
    return maximo;
}
```

```c
int obtener_cantidad_palabras(FILE *archivo, int len){
    int cantidad = 0;
    char c = ' ';
    while (c != EOF){
        c = fgetc(archivo);
        if (c == '\n') cantidad ++;
    }
    return cantidad;
}


void obtener_palabras(FILE* archivo, char **lista_palabras, int len, int
   cantidad_palabras){
    int indice = 0;
    char palabra[len + 1];

    int i;
    for (i = 0; i < cantidad_palabras; i++){
        int j;
        fgets(palabra, len + 1, archivo);
        for(j = 0; j < (strlen(palabra) + 1); j++){
            if (palabra[j] == '\n' || palabra[j] == EOF) palabra[j] = '\0';
        }
        lista_palabras[indice] = malloc(strlen(palabra) + 1);
        strcpy(lista_palabras[indice], palabra);
        indice ++;
    }
}


int comparar_como_numero(const char *numero1,const char *numero2){
    int n1 = atoi(numero1);
    int n2 = atoi(numero2);
    return(n1-n2);
}


void orgaqsortgeneral(char** izq, char** der, int (*fcmp)(const char *,const
    char *)){
    if (izq <= der) {
        char *aux;
        char **inicio = izq;
        char **fin = der;
        while (inicio < fin){
            while ((fcmp(*inicio, *izq) <= 0) && (inicio < der))inicio++;
            while ((fcmp(*fin, *izq) > 0) && (fin > izq)) fin--;
            if (inicio < fin){
                aux = *inicio;
                *inicio = *fin;
                *fin = aux;
            }
        }
        aux = *izq;
        *izq = *fin;
        *fin = aux;
        orgaqsortgeneral(izq, fin-1, fcmp);
        orgaqsortgeneral(fin+1, der, fcmp);
    }
}
```

```
void orgaqsort(char **izq, char **der, int num){
    //si num = 0 ordena alfabeticamente
    //Si num != 0 ordena como numeros
    if (num) orgaqsortgeneral(izq, der, comparar_como_numero);
    else orgaqsortgeneral(izq, der, strcmp);
}


int main(int argc, char *argv[]){

    int longitud, cantidad_palabras, nro_archivo_entrada, nro_archivo_salida
       , criterio_ordenamiento, i;
    char** lista_palabras;
    FILE* archivo;
    FILE* output = stdout;

    if ((argc < 2) || argc > 5) {
        mostrar_error_y_salir(mensaje_error_cantidad_parametros,
           salida_error_parametros);
    }

    else if ((strcmp(argv[1], "-h") == 0) || (strcmp(argv[1], "--help") ==
       0)){
        if(argc != 2) mostrar_error_y_salir(
           mensaje_error_cantidad_parametros, salida_error_parametros);
        mostrar_ayuda();
        return 0;
    }
    else if ((strcmp(argv[1], "-V") == 0) || (strcmp(argv[1], "--version")
       == 0)){
        if(argc != 2) mostrar_error_y_salir(
           mensaje_error_cantidad_parametros, salida_error_parametros);
        mostrar_version();
        return 0;
    }
    // Verificacion para ingreso de ordenamientos y archivos
    else if ((strcmp(argv[1], "-o") == 0) || (strcmp(argv[1], "--output") ==
        0)){
        if (argc != 4) mostrar_error_y_salir(
           mensaje_error_cantidad_parametros, salida_error_parametros);
        nro_archivo_entrada = 3;
        nro_archivo_salida = 2;
        criterio_ordenamiento = 0;
    }

    else if ((argc > 2) && ((strcmp(argv[2], "-o") == 0) || (strcmp(argv[2],
        "--output") == 0)) && ((strcmp(argv[1], "-n") == 0) || (strcmp(argv
       [1], "--numeric") == 0))) {
        if (argc != 5) mostrar_error_y_salir(
           mensaje_error_cantidad_parametros, salida_error_parametros);
        nro_archivo_entrada = 4;
        nro_archivo_salida = 3;
        criterio_ordenamiento = 1;
    }

    else mostrar_error_y_salir(mensaje_error_parametros,
       salida_error_parametros);

    archivo = fopen(argv[nro_archivo_entrada],"r");
    if (!archivo) mostrar_error_y_salir(mensaje_error_archivo_inexistente,
       salida_error_archivo_inexistente);
```

```
    if (strcmp(argv[nro_archivo_salida], "-")){
        output = fopen(argv[nro_archivo_salida], "w");
        if (!output) mostrar_error_y_salir(mensaje_error_archivo_inexistente
            , salida_error_archivo_inexistente);
    }

    longitud = obtener_longitud_maxima(archivo);
    rewind(archivo);
    cantidad_palabras = obtener_cantidad_palabras(archivo, longitud);
    rewind(archivo);
    lista_palabras = malloc(cantidad_palabras * sizeof(char*));
    obtener_palabras(archivo, lista_palabras, longitud, cantidad_palabras);
    orgaqsortassembly(lista_palabras, lista_palabras + cantidad_palabras -1,
        criterio_ordenamiento);
    for (i = 0; i < cantidad_palabras; i++){
        fprintf(output, "%s\n",lista_palabras[i]);
        free(lista_palabras[i]);
    }
    fclose(archivo);
    fclose(output);
    free(lista_palabras);
    return 0;
}
```

4.2. Código Assembly

```
        .text
        .align 2
        .globl orgaqsortassembly
        .ent orgaqsortassembly
orgaqsortassembly:      .set noreorder
        .cpload $25
        .set reorder
        subu    $sp, $sp, 32
        sw      $ra, 24($sp)
        sw      $fp, 20($sp)
        sw      $gp, 16($sp)

        sw      $4, 32($sp)
        sw      $5, 36($sp)
        sw      $6, 40($sp)

        add     $16, $0, $4         # s0 <- izq
        add     $17, $0, $5         # s1 <- der
        add     $20, $0, $6         # s4 <- num
        subu    $9, $17, $16        # if (der <= izq){
        blez    $9, end             # end}

        add     $18, $0, $4         # s2 <- inicio
        add     $19, $0, $5         # s3 <- fin

while1: subu    $10, $18, $19        # if (inicio >= fin) {
        bgez    $10, swap2           #  swap2 }
```

```
while2: subu    $10, $17, $18       # if (inicio >= der){
        blez    $10, while3         #  while3}

        add     $4, $0, $18        # Cargo inicio e izq
        add     $5, $0, $16        # y num para
        add     $6, $0, $20        # comparar
        j       comparar           # Compara *inicio y *izq según num y devuelve en $v0
        blez    $2, aumentar_inicio
aca1:   b      while2

while3: subu    $10, $19, $16       # if(fin <= izq){
        blez    $10, swap1          #   swap1}

        add     $4, $0, $17        # Cargo der para comparar
        add $5, $0, $19
        add $6, $0, $20
        j       comparar
        bgtz    $2, decrementar_fin
aca2:   b      while3

swap1:  lw      $11, 0($18)         #Swap inicio-fin
        lw      $12, 0($19)
        sw      $11, 0($19)
        sw      $12, 0($18)
        b      while1

swap2:  lw      $11, 0($16)         #Swap izq-fin
        lw      $12, 0($19)
        sw      $11, 0($19)
        sw      $12, 0($16)

        addi    $19, $19, -1             # qsort(izq, fin-1, num)
        add     $4, $0, $16
        add     $5, $0, $19
        add     $6, $0, $20
        jal     orgaqsortassembly

        addi    $19, $19, 2              # qsort(fin+1, der, num)
        add     $4, $0, $19
        add     $5, $16, $17
        add     $6, $0, $20
        jal     orgaqsortassembly

end:    lw      $ra, 24($sp)
        lw      $fp, 20($sp)
        lw      $gp, 16($sp)
        addi    $sp, $sp, 32
        jr      $ra

decrementar_fin:    addi   $19, $19, -1
```

```
                    b       aca2

aumentar_inicio:    addi    $18, $18, 1
                    b       aca1
.end    orgaqsortassembly

comparar:   subu    $sp, $sp, 8
            sw      $fp, 4($sp)
            sw      $gp, 0($sp)

            sw      $4, 8($sp)
            sw      $5, 12($sp)
            sw      $6, 16($sp)

            beq     $6, $0, atoi
            lw      $9, 0($4)
            lw      $10, 0($5)
loop:       lb      $11, 0($9)
            lb      $12, 0($10)
            subu    $13, $11, $12
            bltz    $13, dev_menor
            bgtz    $13, dev_mayor
            beq     $11, $0, dev_cero
            b       loop

dev_cero:   add     $2, $0, $0
            b fin

dev_menor:  add     $2, $0, 1
            b fin

dev_mayor:  addi    $2, $0, -1
            b fin

fin:        lw      $fp, 12($sp)
            lw      $gp, 8($sp)
            add     $sp, $sp, 16
            jr      $ra

atoi:       add     $2, $2, $0
            addi    $12, $0, 10     #t4 <-- 10
            move    $8, $4
loop2:      lbu     $9, 0($8)
            beq     $9, $0, fin     #falta fin
            sub     $10, $9, $0
            mul     $2, $2, $12
            add     $2, $2, $12
            addi    $8, $8, 1
            b      loop2
```

```
lw      $fp, 12($sp)
lw      $gp, 8($sp)
jr      $ra
add    $sp, $sp, 16
```