

Trabajo práctico 1: Conjunto de instrucciones MIPS

Menniti, Sebastián Ezequiel - *Padrón 93445*

mennitise@gmail.com

Pérez, Miguel - *Padrón 94708*

miguelangelperez909@gmail.com

Prystupiuk, Maximiliano - *Padrón 94853*

mpzystupiuk@gmail.com

2do. Cuatrimestre de 2018

66.20 Organización de Computadoras

Facultad de Ingeniería, Universidad de Buenos Aires

11 de octubre de 2018

Resumen

El trabajo consiste en ordenar el contenido de un archivo utilizando el algoritmo de ordenamiento "Quicksort". El mismo deberá contener una opción en la cual se le indique si el archivo deberá ordenarse alfabéticamente o numéricamente. Se harán dos versiones de la función quicksort: una en MIPS32 y otra en C (para que el sistema sea portable).



Índice

1. Introducción	3
1.1. Quicksort	3
2. Desarrollo	4
2.1. Diseño e implementación	4
2.2. Herramientas utilizadas	4
2.2.1. Gxemul	4
2.3. Diagrama del stack	4
2.4. Instrucciones de corrida	4
2.4.1. Gxemul	4
3. Código fuente	6
3.1. quicksort.S	6
3.1.1. Descripción	6
3.1.2. Contenido	6
4. Casos de prueba	10
4.1. Prueba 1: Alice	10
4.1.1. Input	10
4.1.2. Resultado esperado	10
4.1.3. Resultado obtenido	11
4.2. Prueba 2: Números 1	12
4.2.1. Input	12
4.2.2. Resultado esperado	12
4.2.3. Resultado obtenido	12
4.3. Prueba 3: Números 2	12
4.3.1. Input	12
4.3.2. Resultado esperado	12
4.3.3. Resultado obtenido	13
4.4. Prueba 4: Números 3	13
4.4.1. Input	13
4.4.2. Resultado esperado	13
4.4.3. Resultado obtenido	13
4.5. Prueba 5: Archivo inexistente	14
4.5.1. Input	14
4.5.2. Resultado esperado	14
4.5.3. Resultado obtenido	14
5. Conclusiones	15



1. Introducción

1.1. Quicksort

QuickSort es un algoritmo de ordenamiento que se basa en la técnica de "divide y vencerás" por la que en cada recursión, el problema se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos.



2. Desarrollo

2.1. Diseño e implementación

El desarrollo fue realizado parte con el lenguaje de programación C y parte en lenguaje ensamblador MIPS. Los programas escritos fueron compilados o ensamblados según el caso, y posteriormente enlazados, utilizando las herramientas de GNU disponibles en el sistema NetBSD utilizado.

2.2. Herramientas utilizadas

2.2.1. Gxemul

GXemul es un emulador de la arquitectura de computadores MIPS. Es software libre bajo una licencia tipo BSD.

2.3. Diagrama del stack

```
1 # STACK
2 #
3 # 48
4 # +-----+
5 # 44 |//////////|
6 # +-----+
7 # 40 |      ra      |
8 # +-----+
9 # 36 |      fp      |
10 # +-----+
11 # 32 |      gp      |
12 # +-----+
13 # 28 |  pivot dir  |
14 # +-----+
15 # 24 |      pivot  |
16 # +-----+
17 # 20 |      right  |
18 # +-----+
19 # 16 |      left   |
20 # +-----+
21 # 12 |//////////|
22 # +-----+
23 # 8  |      a2      |
24 # +-----+
25 # 4  |      a1      |
26 # +-----+
27 # 0  |      a0      |
28 # +-----+
```

2.4. Instrucciones de corrida

2.4.1. Gxemul

1. Dependiendo del entorno en el que se corra, host (Linux) o guest (NetBSD) se ejecutará el make indicando el entorno correspondiente

```
$ make linux
```

```
$ make mips
```



2. Dentro del entorno de NetBSD, en el directorio del proyecto para ordenar, por ejemplo un archivo `filename.txt` y verlo por pantalla, ejecutar:

```
$ ./qsort -o - filename.txt
```

En el caso de quererlo ordenar numericamente, agregar la opcion `-n`, por ejemplo:

```
$ ./qsort -o - -n filename.txt
```

Tambien se puede acceder al manual de ayuda ingresando

```
$ ./qsort -h
```



3. Código fuente

3.1. quicksort.S

3.1.1. Descripción

Archivo con el código fuente de la implementación de algoritmo de ordenamiento Quicksort en el lenguaje ensamblador de MIPS.

3.1.2. Contenido

```
1  #include <mips/regdef.h>
2
3  #define SS 48
4  #define O_RA 40
5  #define O_FP 36
6  #define O_GP 32
7
8  #define ARG_0 48
9  #define ARG_1 52
10 #define ARG_2 56
11
12 #define O_LEFT 16
13 #define O_RIGHT 20
14 #define O_PIVOT 24
15 #define DIR_PIVOT 28
16
17 .text
18 .align      2
19 .globl      qsortMIPS
20 .ent        qsortMIPS
21 # void qsortMIPS(char** left, char** right, int numeric);
22 #  a0 -> left
23 #  a1 -> right
24 #  a2 -> numeric
25
26 qsortMIPS:
27     .set     noreorder
28     .cload   t9
29     .set     reorder
30
31     subu     sp, sp, SS      # Create Stack Frame
32     sw       ra, O_RA(sp)   # Save Return Address
33     sw       $fp, O_FP(sp)
34     sw       gp, O_GP(sp)
35
36     #Cargo los args en el stack
37     sw       a0, ARG_0(sp)  # &left
38     sw       a1, ARG_1(sp)  # &right
39     sw       a2, ARG_2(sp)  # &numeric
40
41     lw       t1, ARG_0(sp)  # t1 = limit_left
42     lw       t2, ARG_1(sp)  # t2 = limit_right
43     lw       t6, ARG_0(sp)  # t6 = left
44     lw       t7, ARG_1(sp)  # t7 = right
45
46     sw       t6, O_LEFT(sp)
47     sw       t7, O_RIGHT(sp)
48
49     # PIVOT - t3 = pivot
```



```
50      # pivot = ( ( t6/4 ) + ( t7/4 ) ) / 2)*4
51      srl      t8, t6, 2
52      srl      t9, t7, 2
53      addu     t3, t8, t9
54      srl      t3, t3, 1
55      sll      t3, t3, 2
56
57      sw       t3, O_PIVOT(sp)
58      lw       t3, 0(t3)
59      sw       t3, DIR_PIVOT(sp)
60      # lbu     t8, 0(t3) # t8 = *pivot
61
62 while_loop:
63     lw       t6, O_LEFT(sp)
64     lw       t7, O_RIGHT(sp)
65     bgtu     t6, t7, end_while_loop # left > right
66
67 while_left_loop:
68     lw       t6, O_LEFT(sp)
69     lw       t2, ARG_1(sp)
70
71     bgeu     t6, t2, while_right_loop # branch (left >= limit_right),
        while_right_loop
72
73     lw       a0, 0(t6)
74     lw       a1, DIR_PIVOT(sp)
75
76     lw       a2, ARG_2(sp)
77     beqz     a2, compare_str_1
78
79 compare_int_1: # compare_int(list[left], pivot)
80     la       t9, compare_int
81     b        compare_call_1
82
83 compare_str_1: # compare_str(list[left], pivot)
84     la       t9, compare_str
85     b        compare_call_1
86
87 compare_call_1:
88     jal      t9
89
90     bgez     v0, while_right_loop
91
92     lw       t6, O_LEFT(sp)
93     addu     t6, t6, 4 # left++
94     sw       t6, O_LEFT(sp)
95     b        while_left_loop
96
97 while_right_loop:
98     lw       t1, ARG_0(sp)
99     lw       t7, O_RIGHT(sp)
100
101     bleu     t7, t1, end_while_left_right_loop # branch (right <= limit_left
        ), end_while_left_right_loop
102
103     lw       a0, DIR_PIVOT(sp)
104     lw       a1, 0(t7)
105
106     lw       a2, ARG_2(sp)
```



```
107     beqz    a2, compare_str_2
108
109 compare_int_2: # compare_int(pivot, list[right])
110     la      t9, compare_int
111     b       compare_call_2
112
113 compare_str_2: # compare_str(pivot, list[right])
114     la      t9, compare_str
115     b       compare_call_2
116
117 compare_call_2:
118     jal     t9
119
120     bgez    v0, end_while_left_right_loop
121
122     lw      t7, O_RIGHT(sp)
123     subu    t7, t7, 4 # right—
124     sw      t7, O_RIGHT(sp)
125     b       while_right_loop
126
127 end_while_left_right_loop:
128     lw      t6, O_LEFT(sp)
129     lw      t7, O_RIGHT(sp)
130     bgtu    t6, t7, while_loop # left > right
131
132 swap:
133     lw      t4, 0(t6)
134     lw      t5, 0(t7)
135     sw      t5, 0(t6)
136     sw      t4, 0(t7)
137
138     lw      t6, O_LEFT(sp)
139     addu    t6, t6, 4 # left++
140     sw      t6, O_LEFT(sp)
141     lw      t7, O_RIGHT(sp)
142     subu    t7, t7, 4 # right—
143     sw      t7, O_RIGHT(sp)
144     b       while_loop
145
146 end_while_loop:
147     sw      t6, O_LEFT(sp)
148     sw      t7, O_RIGHT(sp)
149
150 qsort_left:
151     lw      t1, ARG.0(sp)
152     lw      t7, O_RIGHT(sp)
153     bgeu    t1, t7, qsort_right # limit_left >= right
154
155     lw      a0, ARG.0(sp) # a0 - limit_left
156     lw      a1, O_RIGHT(sp) # a1 - right
157     la      t9, qsortMIPS
158     jal     t9
159     lw      a0, ARG.0(sp)
160     lw      a1, ARG.1(sp)
161
162 qsort_right:
163     lw      t2, ARG.1(sp)
164     lw      t6, O_LEFT(sp)
165     bleu    t2, t6, _exit # limit_right <= left
```




```
166
167     lw      a0, O_LEFT(sp) # a0 – left
168     lw      a1, ARG_1(sp) # a1 – limit_right
169     la      t9, qsortMIPS
170     jal     t9
171     lw      a0, ARG_0(sp)
172     lw      a1, ARG_1(sp)
173
174 _exit:
175     #Desarmo el Stack Frame
176     lw      ra, O_RA(sp)
177     lw      $fp, O_FP(sp)
178     lw      gp, O_GP(sp)
179     addu    sp, sp, SS
180     j       ra
181
182 .end qsortMIPS
```



4. Casos de prueba

4.1. Prueba 1: Alice

4.1.1. Input

```
1 ./qsort -o - alice.txt
```

4.1.2. Resultado esperado

```
1 "and
2 "without
3 ALICE
4 Alice ,
5 a
6 and
7 bank,
8 beginning
9 book
10 book, "
11 but
12 by
13 conversations
14 conversations?"
15 do:
16 get
17 had
18 had
19 having
20 her
21 her
22 in
23 into
24 is
25 it
26 it ,
27 no
28 nothing
29 of
30 of
31 of
32 on
33 once
34 or
35 or
36 or
37 peeped
38 pictures
39 pictures
40 reading ,
41 she
42 sister
43 sister
44 sitting
45 the
46 the
47 the
48 thought
```



49 tired
50 to
51 to
52 twice
53 use
54 very
55 was
56 was
57 what

4.1.3. Resultado obtenido

1 "and
2 "without
3 ALICE
4 Alice ,
5 a
6 and
7 bank,
8 beginning
9 book
10 book, "
11 but
12 by
13 conversations
14 conversations?"
15 **do:**
16 get
17 had
18 had
19 having
20 her
21 her
22 in
23 into
24 is
25 it
26 it ,
27 no
28 nothing
29 of
30 of
31 of
32 on
33 once
34 or
35 or
36 or
37 peeped
38 pictures
39 pictures
40 reading ,
41 she
42 sister
43 sister
44 sitting
45 the
46 the



```
47 the
48 thought
49 tired
50 to
51 to
52 twice
53 use
54 very
55 was
56 was
57 what
```

4.2. Prueba 2: Números 1

4.2.1. Input

```
1 ./qsort -o - numeros.txt
```

4.2.2. Resultado esperado

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
```

4.2.3. Resultado obtenido

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
```

4.3. Prueba 3: Números 2

4.3.1. Input

```
1 ./qsort -o - numeros_2.txt
```

4.3.2. Resultado esperado

```
1 10
2 11
3 12
4 5
5 6
6 7
```



```
7 8
8 9
9 aaa
10 abc
```

4.3.3. Resultado obtenido

```
1 10
2 11
3 12
4 5
5 6
6 7
7 8
8 9
9 aaa
10 abc
```

4.4. Prueba 4: Números 3

4.4.1. Input

```
1 ./qsort -o - -n numeros_2.txt
```

4.4.2. Resultado esperado

```
1 5
2 6
3 7
4 8
5 9
6 10
7 11
8 12
9 aaa
10 abc
```

4.4.3. Resultado obtenido

```
1 5
2 6
3 7
4 8
5 9
6 10
7 11
8 12
9 aaa
10 abc
```



4.5. Prueba 5: Archivo inexistente

4.5.1. Input

1 `./qsort -o - -n nothing.txt`

4.5.2. Resultado esperado

1 No such file

4.5.3. Resultado obtenido

1 No such file



5. Conclusiones

Para los sets de pruebas pasados el tiempo de ejecución es similar entre las versiones C y MIPS. Pasandole archivos cada vez más grandes se hace notoria la diferencia de velocidad entre ambas implementaciones, siendo la más rápida la de MIPS.