

嵌入式系统设计方法 实验 3

PB18111697 王章瀚

2020 年 12 月 12 日

目录

1 实验要求概述	2
2 实验过程	2
2.1 输入输出	2
2.2 排序函数	2
2.2.1 字符串比较函数	2
2.2.2 插入排序函数	3
3 汇编代码和 C 代码混合编译	5
4 代码运行	6
5 总结	7

1 实验要求概述

- 用 C 语言完成字符串的输入输出
- 用 ARM 汇编语言完成排序操作

2 实验过程

2.1 输入输出

既然要求用 C 语言输入输出, 那么这很自然地能写出以下代码:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define STRING_NUM 7
5 #define STRING_LENGTH 128
6
7 extern void strsort(char** strs, int n);
8
9 int main() {
10     char* strs[STRING_NUM];
11     int i;
12     for (i = 0; i < STRING_NUM; i++) {
13         strs[i] = (char*)malloc(sizeof(char) * STRING_LENGTH);
14     }
15
16     for (i = 0; i < STRING_NUM; i++) {
17         scanf("%s", strs[i]);
18     }
19
20     strsort(strs, STRING_NUM);
21
22     for (i = 0; i < STRING_NUM; i++) {
23         printf("%s\n", strs[i]);
24     }
25 }
```

这里的 `strsort` 是待使用 ARM 汇编完成的字符串排序代码.

2.2 排序函数

2.2.1 字符串比较函数

字符串比较函数这里主要是有一个循环, 用这个循环逐字符地比较大小, 直到出现不相等的情况, 就可以得出两个字符串的大小关系. 其代码及注释 (还有概述) 如下:

```

1  .global strcmp
2  .type   strcmp, %function
3  strcmp:
4      str fp, [sp, #-4]! @ 把 fp 暂存一下
5      add fp, sp, #0     @ fp = sp
6      sub sp, sp, #20    @ 扩栈, sp -= 20
7      str r0, [fp, #-16] @ 暂存参数 fp[-16] = str1
8      str r1, [fp, #-20] @ 暂存参数 fp[-20] = str2
9      mov r3, #0         @ r3 = 0
10     str r3, [fp, #-8]   @ fp[-8] = r3
11  .cmp_loop:
12     ldr r3, [fp, #-8]   @ r3 = i
13     ldr r2, [fp, #-16]  @ r2 = str1
14     add r3, r2, r3      @ r3 = &(str1[i])
15     ldrb r2, [r3]       @ r2 = str1[i]
16     ldr r3, [fp, #-8]   @ r3 = i
17     ldr r1, [fp, #-20]  @ r1 = str2
18     add r3, r1, r3      @ r3 = &(str2[i])
19     ldrb r3, [r3]       @ r3 = str2[i]
20     cmp r2, r3          @ 比较 str1[i] 和 str2[i]
21     bne .cmp_done      @ 如果 str1[i] != str2[i] 就结束 while, 跳到 .L3
22     cmp r2, #0         @ 比较 str1[i] 和 0
23     beq .cmp_done      @ 如果 str1[i] == 0, 就继续循环
24     ldr r3, [fp, #-8]   @
25     add r3, r3, #1      @ i++
26     str r3, [fp, #-8]   @
27     b .cmp_loop
28  .cmp_done:
29     sub r0, r2, r3      @ 计算返回值 str1[i] - str2[i]
30     add sp, fp, #0      @ sp = fp
31     @ sp needed
32     ldr fp, [sp], #4    @ 放回 fp
33     bx lr              @ 返回
34     .size   strcmp, .-strcmp
35     .align  2

```

- 4-10 行代码是一些函数调用的准备操作
- 11-27 行代码是循环体的比较过程，当比较结束，就跳转到.cmp_done
- 28-33 行代码就是设置返回值，并返回

2.2.2 插入排序函数

插入排序的算法这里就不赘述了，下面上代码和讲解。

- 首先是进入函数，并做一些进入函数该做的事。

```

37  @ strsort 函数
38  .globl  strsort
39  .p2align 2
40  .type   strsort, %function
41  strsort:
42  .fnstart
43      push {fp, lr}      @ 暂存 fp, lr(帧指针, 返回地址)
44      add fp, sp, #4     @ fp = sp + 4
45      sub sp, sp, #24    @ sp = sp - 24, 扩栈
46      str r0, [fp, #-24] @ fp[-24] = str
47      str r1, [fp, #-28] @ fp[-28] = n
48      mov r3, #1        @ i = r3 = 1
49      str r3, [fp, #-12] @ fp[-12] = i
50      b .outer_loop     @ 跳转到外层 for 循环

```

- 主要做了一些函数调用的入栈和参数保存等操作。

- 然后是外层循环：

```

37 @ strsort 函数
38 .globl strsort
39 .p2align 2
40 .type strsort, %function
41 strsort:
42 .fnstart
43 push {fp, lr} @ 暂存 fp, lr(帧指针, 返回地址)
44 add fp, sp, #4 @ fp = sp + 4
45 sub sp, sp, #24 @ sp = sp - 24, 扩栈
46 str r0, [fp, #-24] @ fp[-24] = str
47 str r1, [fp, #-28] @ fp[-28] = n
48 mov r3, #1 @ i = r3 = 1
49 str r3, [fp, #-12] @ fp[-12] = i
50 b .outer_loop @ 跳转到外层 for 循环
51 .outer_prepare:
52 ldr r3, [fp, #-12] @ j = r3 = i
53 sub r3, r3, #1 @ j = r3 = i - 1
54 str r3, [fp, #-8] @ fp[-8] = j
55 ldr r3, [fp, #-12] @ r3 = i
56 lsl r3, r3, #2 @ r3 = i << 2
57 ldr r2, [fp, #-24] @ r2 = str
58 add r3, r2, r3 @ r3 = &(strs[i])
59 ldr r3, [r3] @ r3 = strs[i]
60 str r3, [fp, #-16] @ fp[-16] = strs[i]
61 b .inner_loop @ 进入内层 while 循环
62
63 > .pushback: ...
78 > .inner_loop: ...
91 > .put_i: @ 否则就内存循环结束 ...
103 .outer_loop:
104 ldr r2, [fp, #-12] @ r2 = i
105 ldr r3, [fp, #-28] @ r3 = n
106 cmp r2, r3 @ 比较 i 和 n
107 blt .outer_prepare @ 若 i < n, 跳转到 .L11
108 nop
109 sub sp, fp, #4 @ 恢复 sp
110 @ sp needed
111 pop {fp, pc} @ 恢复 fp, pc
112 .size strsort, .-strsort
113 .align 2
114
115 .fnend

```

- 中间内层循环暂时缩起来了
- 可以看到外层循环先进入.outer_loop 来判断循环结束条件 (返回值部分也在这里)
- 而.outer_prepare 部分则是保存了 strs[i] 的值 (这里 strs 是字符串数组)

- 再说内层循环:

```

63 .pushback:
64     ldr r3, [fp, #-8]    @ r3 = j
65     lsl r3, r3, #2       @ r3 = j << 2
66     ldr r2, [fp, #-24]   @ r2 = str
67     add r2, r2, r3       @ r2 = &(str[j])
68     ldr r3, [fp, #-8]    @ r3 = j
69     add r3, r3, #1       @ r3 = j + 1
70     lsl r3, r3, #2       @ r3 = (j + 1) << 2
71     ldr r1, [fp, #-24]   @ r1 = str
72     add r3, r1, r3       @ r3 = &(str[j+1])
73     ldr r2, [r2]         @ r2 = str[j]
74     str r2, [r3]         @ str[j+1] = str[j]
75     ldr r3, [fp, #-8]    @ r3 = j
76     sub r3, r3, #1       @ j--
77     str r3, [fp, #-8]    @ fp[-8] = j
78 .inner_loop:
79     ldr r3, [fp, #-8]    @ r3 = j
80     cmp r3, #0           @ 比较 j 和 0
81     blt .put_i          @ 若 j < 0, 内层循环结束, 跳转到 .L9
82     lsl r3, r3, #2       @ j << 2
83     ldr r2, [fp, #-24]   @ r2 = str
84     add r3, r2, r3       @ r3 = &(str[j])
85     ldr r3, [r3]         @ r3 = str[j]
86     ldr r1, [fp, #-16]   @ r1 = str[i]
87     mov r0, r3           @ r0 = str[j]
88     bl strcmp           @ strcmp(str[j], str[i])
89     cmp r0, #0           @ 比较 strcmp(str[j], str[i]) 和 0
90     bgt .pushback       @ 如果 str[j] > str[i], 则需要让str[j+1] = str[j]
91 .put_i:
92     ldr r3, [fp, #-8]    @ r3 = j
93     add r3, r3, #1       @ r3 = j + 1
94     lsl r3, r3, #2       @ r3 = (j+1) << 2
95     ldr r2, [fp, #-24]   @ r2 = str
96     add r3, r2, r3       @ r2 = &(str[j+1])
97     ldr r2, [fp, #-16]   @ r2 = str[i]
98     str r2, [r3]         @ str[j+1] = str[i]
99     ldr r3, [fp, #-12]   @ r3 = i
100    add r3, r3, #1       @ i++
101    str r3, [fp, #-12]   @ fp[-12] = i

```

- 内层循环中, `.inner_loop` 用来判断循环继续的条件 (也就是 `strs[j] > str[i]`), 如果依然成立, 就做 `.push_back`
- `.push_back` 是把当前 `strs[j+1] = str[j]`, 实现后推
- 如果循环结束了, 就把 `strs[i]` 放到循环结束了的 `j+1` 位置.

- 以上过程完全是插入排序的过程, 只不过用汇编表示出来了. 而其中代码块的含义我已经写出来, 每条代码我也给予了一定注释. 我想应该足够清楚了吧.

3 汇编代码和 C 代码混合编译

有了汇编代码和 C 代码, 接下来的问题就是如何混合编译起来形成可执行文件.

这其实很简单, 只需要编译好汇编代码, 生成相应的.o 文件, 再和 C 代码一起传给 gcc 来编译即可. 为了方便, 我写了一个 Makefile, 如下:

```

1 PLAT=arm-linux-
2 AS=$(PLAT) as
3 CC=$(PLAT) gcc
4
5 sort: main.c strsort.o
6     $(CC) main.c strsort.o -o sort
7
8 strsort: strsort.s
9     $(AS) strsort.s -o strsort.o

```

用这样一个 Makefile, 就能很轻易地生成对应可执行文件, 然后用串口传给开发板.

4 代码运行

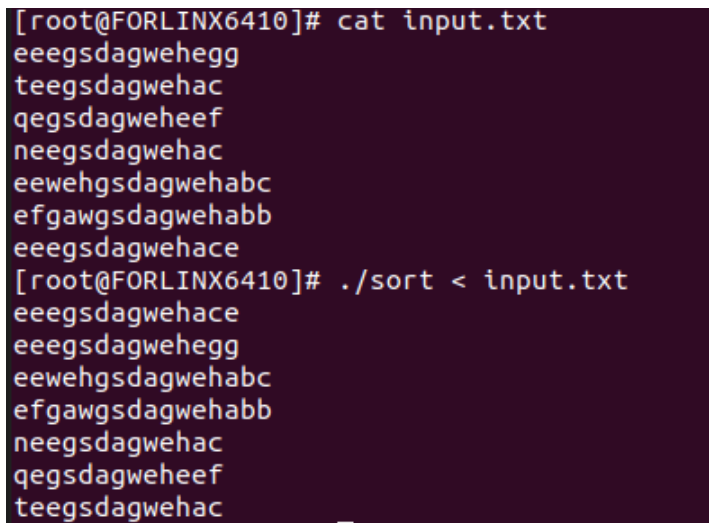
精心挑选了一个这样一些字符串

```
1 eeegsdagwehegg
2 teegsdagwehac
3 qegsdagweheef
4 neegsdagwehac
5 eewehgsdagwehac
6 efgawgsdagwehabb
7 eeegsdagwehace
```

其排序结果如下:

```
1 eeegsdagwehace
2 eeegsdagwehegg
3 eewehgsdagwehac
4 efgawgsdagwehabb
5 neegsdagwehac
6 qegsdagweheef
7 teegsdagwehac
```

代码运行截图:



```
[root@FORLINX6410]# cat input.txt
eeegsdagwehegg
teegsdagwehac
qegsdagweheef
neegsdagwehac
eewehgsdagwehac
efgawgsdagwehabb
eeegsdagwehace
[root@FORLINX6410]# ./sort < input.txt
eeegsdagwehace
eeegsdagwehegg
eewehgsdagwehac
efgawgsdagwehabb
neegsdagwehac
qegsdagweheef
teegsdagwehac
```

5 总结

本实验用 ARM 汇编完成了一个插入排序, 并和 C 代码混合使用. 收获颇丰.