

0926_综合研究4研究报告

摘要

源程序文件（.c文件）需要进行编译、连接两步工作后生成 `exe` 文件在前面的内容中，这两步工作是集成在一起完成的。

多个 `obj` 文件中的代码可以存储在一个 `lib` 文件中，对于 `tc2.0` 开发环境，一个 `exe` 文件中可能包含了来自多个 `obj` 文件和 `lib` 文件中的代码。

`cs.lib` `emu.lib` `maths.lib` 等 `lib` 文件中存储着C语言库函数的代码，比如 `printf`、`getch` 等等C语言提供的库函数都在 `cs.lib` 中存储。

我们用形如 `tcc a.c` 的方法对程序进行编译连接，使用的是 `tc2.0` 设计的一套固定的生成 `exe` 文件的方案。这套设计好的方案的具体步骤如下：

- (1) `tcc` 将源程序文件编译为 `a.obj`
- (2) `tcc` 调用 `tlink` 将 `c0s.obj`、`cs.lib`、`emu.lib` `maths.lib` 中的 `a.obj` 中的程序要用到的代码与 `a.obj` 的代码连接到一起生成 `exe` 文件。

而来自 `c0s.obj` 中的代码被连接到其他代码的前面。`c0s.obj` 中的代码所做的工作是：进行相关的初始化工作、调用名称为“`main`”的函数、其他工作。

因为 `c0s.obj` 的代码被连接到其他代码前面，则 `exe` 文件运行的时候首先运行来自 `c0s.obj` 中的代码，进行相关的初始化工作，然后调用 `main` 函数，从此开始运行程序员写的程序。

我们可以看出，这套工作方案落实了C语言的“用户程序必须从 `main` 函数开始”的规则。

问题研究

1. 用 `tcc` 将下面的程序编译为 `obj` 文件。

```
1 int f(void) { return 1; }
```

| | | |
|---|-------|---|
| C | f.c | U |
| U | F.OBJ | U |

通过查看 `tcc` 的使用方式可以看到 `-c` 为生成 `obj` 文件的参数

```
-K      Default char is unsigned      -Lxxx   Libraries directory
-M      Generate link map              -N      Check stack overflow
-O      Optimize jumps                 -S      Produce assembly output
-Uxxx   Undefine macro                 -Z      Optimize register usage
-a      Generate word alignment        -c      Compile only
-d      Merge duplicate strings        -xxxx   Executable file name
-f      * Floating point emulator      -f87    8087 floating point
-gN     Stop after N warnings          -iN     Maximum identifier length N
-jN     Stop after N errors            -k      Standard stack frame
-lx     Pass option x to linker        -mc     Compact Model
-mh     Huge Model                    -ml     Large Model
-mm     Medium Model                  -ms     * Small Model
-mt     Tiny Model                    -nxxx   Output file directory
-oxxx   Object file name              -p      Pascal calls
-r      * Register variables           -u      * Underscores on externs
-v      Source level debugging         -w      Enable all warnings
-wxxx   Enable warning xxx            -w-xxx  Disable warning xxx
-y      Produce line number info       -zxxx   Set segment names

C:\>tcc -nsrc\four -c \SRC\FOUR\F.C
Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International
\src\four\f.c:

Available memory 458420
```

2. 用 `tcc` 的方法编译连接下面的程序。注意显示出来的信息。这些信息说明了什么？

```
1  main() { f(); }
```

通过编译链接，显示出来没有 `f` 的定义，说明 `tlink` 链接时仅仅链接 `c0s.obj`、`cs.lib`、`emu.lib` `maths.lib` 其余的用户自定义的不会被自动链接。

```
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c F:\gitee\ThreeOneProject\31prj_c
Drive C is mounted as local directory F:\gitee\ThreeOneProject\31prj_c\

Z:\>c:

C:\>tcc -nsrc\four \SRC\FOUR\A.C
Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International
\src\four\A.C:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
Undefined symbol '_f' in module A.C

Available memory 458466

C:\>
```

3. `tc2.0` 提供一个工具 `tlib.exe`，可以用 `tib.exe` 将一个 `obj` 文件中的代码加到一个 `lib` 文件中。

找到 `tlib.exe`，研究它的使用方法，将（1）中生成的 `obj` 文件加入到 `csib` 中
注意:我们要对一个对象（`cs.lib`）进行正确的改动，但是种正确的改动不一定一次成功
所以，在改动之前，我们可以将原来的对象保存一份，以便恢复。
上面的工作成功后，用 `tcc a.c` 的方法将程序 `c` 编译连接为 `a.exe` 文件用 `debug` 加载 `a.exe` 文件，找到 `main` 函数和 `f` 函数的代码。

问题: `a.c` 中并没有写函数 `f`，`a.exe` 中的函数 `f` 的代码是在什么时候加入的？

- 通过观察 `tlib.exe` 的使用方式后把 `f.obj` 添加到 `cs.lib` 中（提前备份以防止失败）后编译链接 `a.c` 生成可执行文件

```
C:\>TLIB.EXE CS.LIB +\src\four\f.obj
TLIB Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\>tcc -nsrc\four \SRC\FOUR\A.C
Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International
\src\four\a.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

Available memory 458466
```

可以看到链接过程没有报错。

- 通过 `debug` 来观察 `a.exe` 可以看到程序开始后调用子程序然后我们开始转到子程序观察发现子程序为 `f`。

```
-u 01fa
076A:01FA E85703      CALL    0554      main
076A:01FD C3          RET
076A:01FE C3          RET
-u 0554
076A:0554 BB0100      MOV     AX,0001
076A:0557 EB00      JMP     0559      f()
076A:0559 C3          RET
```

- 在问题2中在链接的时候报错而修改完 `cs.lib` 后在进行编译链接正常故 `f` 中的代码是在链接的时候加入到 `a.exe` 中的。

4. 程序b.c中并没有写f、f2和 printf函数，bexe中这些函数的代码是什么时候加

将下面的程序编译为 `f.obj`，将 `f.obj` 加入 `cs.lib`

程序 `f.c`

```
1  int f1(int a, int b) {
2      int c;
3      c = a + b;
4      return c;
5  }
6  int f2(int a, int b) {
7      int c;
8      c = a - b;
9      return c;
10 }
11 int f3(int a, int b) { return a + b + 1; }
```

- 通过 `tcc.exe`, `tlib.exe` 将生成后 `obj` 文件添加到 `cs.lib` 中

```
C:\>tcc -nSRC\FOUR -c \SRC\FOUR\F.C
Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International
\src\four\f.c:

Available memory 458102

C:\>TLIB.EXE CS.LIB +\SRC\FOUR\F.OBJ
TLIB Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\>_
```

将下面的程序编译连接为 `b.exe` 用 `debug` 加载 `b.exe`，找到其中所有函数代码。

程序 `b.c`

```

1  int func(int, int);
2
3  int a, b;
4
5  main() {
6      a = f1(1, 2);
7      b = f2(20, 10);
8      a = func(a, b);
9      printf("%d\n", a);
10 }
11
12 int func(int a, int b) { return a * b; }

```

- 通过 `tcc` 编译链接生成 `b.exe` 可以找到函数代码

```

-u 01fa
076A:01FA B80200      MOV     AX,0002
076A:01FD 50              PUSH    AX
076A:01FE B80100      MOV     AX,0001
076A:0201 50              PUSH    AX
076A:0202 E89411      CALL    1399
076A:0205 59              POP     CX
076A:0206 59              POP     CX
076A:0207 A32604      MOV     [0426],AX
076A:020A B80A00      MOV     AX,000A
076A:020D 50              PUSH    AX
076A:020E B81400      MOV     AX,0014
076A:0211 50              PUSH    AX
076A:0212 E89511      CALL    13AA
076A:0215 59              POP     CX
076A:0216 59              POP     CX
076A:0217 A32804      MOV     [0428],AX
-u 1399
076A:1399 55              PUSH    BP
076A:139A 8BEC      MOV     BP,SP
076A:139C 56              PUSH    SI
076A:139D 8B7604      MOV     SI,[BP+04]
076A:13A0 037606      ADD     SI,[BP+06]
076A:13A3 8BC6      MOV     AX,SI
076A:13A5 EB00      JMP     13A7
076A:13A7 5E              POP     SI
076A:13A8 5D              POP     BP
076A:13A9 C3              RET
-u 13aa
076A:13AA 55              PUSH    BP
076A:13AB 8BEC      MOV     BP,SP
076A:13AD 56              PUSH    SI
076A:13AE 8B7604      MOV     SI,[BP+04]
076A:13B1 2B7606      SUB     SI,[BP+06]
076A:13B4 8BC6      MOV     AX,SI
076A:13B6 EB00      JMP     13B8
076A:13B8 5E              POP     SI
076A:13B9 5D              POP     BP
076A:13BA C3              RET
-u 0238
076A:0238 55              PUSH    BP
076A:0239 8BEC      MOV     BP,SP
076A:023B 8B4604      MOV     AX,[BP+04]
076A:023E F76606      MUL     WORD PTR [BP+06]
076A:0241 EB00      JMP     0243
076A:0243 5D              POP     BP
076A:0244 C3              RET

```

b.exe 中的函数代码是链接时加入的，并且其中包含 f3 的代码紧紧跟在 f2 的后面，猜测加入代码是通过 obj 为单位进行添加的

```
-u 13aa
076A:13AA 55          PUSH    BP
076A:13AB 8BEC          MOV     BP,SP
076A:13AD 56          PUSH    SI
076A:13AE 8B7604        MOV     SI,[BP+04]
076A:13B1 2B7606        SUB     SI,[BP+06]
076A:13B4 8BC6          MOV     AX,SI
076A:13B6 EB00          JMP     13B8
076A:13B8 5E          POP     SI
076A:13B9 5D          POP     BP
076A:13BA C3          RET
076A:13BB 55          PUSH    BP
076A:13BC 8BEC          MOV     BP,SP
076A:13BE 8B4604        MOV     AX,[BP+04]
076A:13C1 034606        ADD     AX,[BP+06]
076A:13C4 40          INC     AX
076A:13C5 EB00          JMP     13C7
076A:13C7 5D          POP     BP
076A:13C8 C3          RET
```

现在尝试将 f.c 中的三个函数分成三份然后逐步添加到 cs.lib 中。

```
-u 13aa
076A:13AA 55          PUSH    BP
076A:13AB 8BEC          MOV     BP,SP
076A:13AD 56          PUSH    SI
076A:13AE 8B7604        MOV     SI,[BP+04]
076A:13B1 2B7606        SUB     SI,[BP+06]
076A:13B4 8BC6          MOV     AX,SI
076A:13B6 EB00          JMP     13B8
076A:13B8 5E          POP     SI
076A:13B9 5D          POP     BP
076A:13BA C3          RET
076A:13BB 0000          ADD     [BX+SI],AL
076A:13BD 0000          ADD     [BX+SI],AL
076A:13BF 0000          ADD     [BX+SI],AL
076A:13C1 0000          ADD     [BX+SI],AL
076A:13C3 005475        ADD     [SI+75],DL
076A:13C6 7262          JB      142A
```

可以看到 f2 后面就没有 f3 的代码了

5. 用 tlib 将 cs.lib 中 printf 函数的代码变为下面的程序的代码:

```
1  printf() { puts("Do you want to use printf? No printf here."); }
```

使得调用 printf 的用户程序，比如:

```
1  main() {
2      int a, b;
3      a = 1;
4      b = 2;
5      printf("%d\n", a + b);
6  }
```

在用 tcc 编译连接后，运行时打印出 Do you want to use printf No printf here

1. 将 printf 从 cs.lib 中移除

```
C:\>TLIB.EXE CS.LIB -printf
TLIB Version 2.0 Copyright (c) 1987, 1988 Borland International
C:\>_
```

2. 将 `printf` 生成新的 `obj` 文件后加入 `cs.lib` 中

```
C:\>tcc -nsrc\four -c \SRC\FOUR\PRINTF.C
Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International
\src\four\printf.c:

    Available memory 457402

C:\>TLIB.EXE CS.LIB +\SRC\FOUR\PRINTF.OBJ
TLIB Version 2.0 Copyright (c) 1987, 1988 Borland International
```

3. 编译链接运行 `main.c` 可以看到结果

```
C:\>tcc -nsrc\four \SRC\FOUR\MAIN.C
Turbo C Version 2.0 Copyright (c) 1987, 1988 Borland International
\src\four\main.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

    Available memory 457382

C:\>\src\FOUR\MAIN.EXE
Do you want to use printf? No printf here.
```