0925_综合研究3研究报告

研究1

摘要

> 用 `debug` 对一下程序编译链接后的代码进行研究

```
unsigned int n;
void f1();
int f2(int, int);
void far f3();

main() {
    n = 0;
    f1();
    n = f2(1, 2);
    f3();
}

void f1() {
    n = 1;
}
int f2(int a, int b) {
    int c;
    c = a + b;
    return c;
}

void far f3() {
    n = 10;
}
```

问题

1. 程序运行时n、a、b、c的段地在哪个存器中?
2. 全局变量的存空间在什么段里?局都量的储空间在什么段里?参数的存储空间在什么段里?函数的返回在什么地方?
3. 全局变量的存空间在什么时候分配?什么时候释?
4. 局部变量的存空间在什么时候分配?什么时候释?
5. 参数的存储空间在什么时候分配?什么时候释?
6. 函数3在调用与返回方式上与函数与2有何不同?

研究过程

- 首先通过 `TCC` 编译链接生成汇编文件

```
        ifndef  ??version
 ?debug  macro
        endm
```

```
    endif
    ?debug  S "\src\three.c"
_TEXT   segment byte public 'CODE'
DGROUP  group   _DATA,_BSS
    assume  cs:_TEXT,ds:DGROUP,ss:DGROUP ;数据段地址和栈段地址为同一块内存空间
_TEXT   ends
_DATA   segment word public 'DATA'
d@  label   byte
d@w label   word
_DATA   ends
_BSS    segment word public 'BSS'
b@  label   byte
b@w label   word
    ?debug  C E9826239510C5C7372635C74687265652E63
_BSS    ends
_TEXT   segment byte public 'CODE'
;   ?debug  L 10
_main   proc    near
;   ?debug  L 11
    mov word ptr DGROUP:_n,0 ;变量n
;   ?debug  L 12
    call    near ptr _f1
;   ?debug  L 13
    mov ax,2
    push    ax
    mov ax,1
    push    ax
    call    near ptr _f2
    pop cx
    pop cx
    mov word ptr DGROUP:_n,ax
;   ?debug  L 14
    call    far ptr _f3
@1:
;   ?debug  L 15
    ret
_main   endp
;   ?debug  L 17
_f1 proc    near
;   ?debug  L 17
    mov word ptr DGROUP:_n,1
@2:
;   ?debug  L 17
    ret
_f1 endp
;   ?debug  L 18
_f2 proc    near
    push    bp
    mov bp,sp
    push    si
;   ?debug  L 20
    mov si,word ptr [bp+4]
    add si,word ptr [bp+6]
;   ?debug  L 21
    mov ax,si
    jmp short @3
@3:
;   ?debug  L 22
```

```
62      pop si
63      pop bp
64      ret
65  _f2 endp
66  ;   ?debug  L 24
67  _f3 proc    far
68  ;   ?debug  L 24
69      mov word ptr DGROUP:_n,10
70  @4:
71  ;   ?debug  L 24
72      ret
73  _f3 endp
74  _TEXT   ends
75  _BSS    segment word public 'BSS'
76  _n  label   word
77      db  2 dup (?)
78  _BSS    ends
79      ?debug  C E9
80  _DATA   segment word public 'DATA'
81  s@  label   byte
82  _DATA   ends
83  _TEXT   segment byte public 'CODE'
84  _TEXT   ends
85      public  _main
86      public  _n
87      public  _f3
88      public  _f2
89      public  _f1
90      end
91  .
```

- 观察汇编代码可以发现变量 `n` 的段地址寄存器为 `ds` ，而 `a` , `b` 的段地址寄存器为 `ss` ， ~~`c` 的段地址寄存器为 `cs` 。~~ ， `c` 的段地址寄存器为 `ss`
- 在此程序中 `n` 为全局变量并且存放在数据段中故全局变量存放在数据段中。 `a,b` 为参数存放在栈中过参数存放在栈中，
- 全局变量的存储空间在编译的时候就分配好有多大然后在执行可执行文件时在分配内存。
- 局部变量在程序执行过程中用到的时候在分配，
- 参数在给函数传递前分配空间。
- `f3` 和 `f2` 的调用方式可以通过汇编代码观察到 `call far ptr _f3` 和 `call near ptr _f2` 一个是 `call fa` 一个是 `` ` call near `` 而返回时 `f2` 先进行栈恢复然后在返回， `f3` 直接返回不进行栈恢复。

> 研究全部完成后，可以回答第2个问题

- 全局变量存储在数据段中
- 局部变量存储在栈中
- 参数存储在栈中
- 返回值存储在寄存器中或者数据段中

研究2

摘要

```c
void f(void);
main() {
    f();
    f();
}
void f(void) {
    int n = 0;
    static int a = 0;
    n++;
    a++;
    printf("  %d  %d\n", n, a);
}
```

问题

- 变量 `n` 与 `a` 的存储空间分配方式何不同?

研究过程

- 通过 `tcc` 生成汇编文件。

```
    ifndef  ??version
?debug  macro
    endm
    endif
    ?debug  S "\src\three2.c"
_TEXT   segment byte public 'CODE'
DGROUP  group   _DATA,_BSS
    assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT   ends
_DATA   segment word public 'DATA'
d@  label   byte
d@w label   word
_DATA   ends
_BSS    segment word public 'BSS'
b@  label   byte
b@w label   word
    ?debug  C E9A96A39510D5C7372635C7468726565322E63
_BSS    ends
_TEXT   segment byte public 'CODE'
;   ?debug  L 2
_main   proc    near
;   ?debug  L 3
    call    near ptr _f
;   ?debug  L 4
    call    near ptr _f
@1:
;   ?debug  L 5
    ret
_main   endp
_TEXT   ends
_DATA   segment word public 'DATA'
    dw  0
_DATA   ends
_TEXT   segment byte public 'CODE'
```

```
;   ?debug  L 6
_f  proc    near
    push    si
;   ?debug  L 7
    xor si,si
;   ?debug  L 9
    inc si
;   ?debug  L 10
    inc word ptr DGROUP:d@
;   ?debug  L 11
    push    word ptr DGROUP:d@
    push    si
    mov ax,offset DGROUP:s@
    push    ax
    call    near ptr _printf
    add sp,6
@2:
;   ?debug  L 12
    pop si
    ret
_f  endp
_TEXT   ends
    ?debug  C E9
_DATA   segment word public 'DATA'
s@  label   byte
    db  32
    db  32
    db  37
    db  100
    db  32
    db  32
    db  37
    db  100
    db  10
    db  0
_DATA   ends
_TEXT   segment byte public 'CODE'
    extrn   _printf:near
_TEXT   ends
    public  _main
    public  _f
    end
·
```

- 观察可以发现 `n` 时局部变量分配在栈空间中而 `a` 分配在数据段中

研究3

摘要

```c
unsigned int a = 1;
unsigned int b = 1;
unsigned char c = 1;
unsigned int a1 = 1;
unsigned long a2 = 1;

main() {
    a++;
    b++;
    c++;
    a1++;
    a2++;
}
```

问题

1. 程序中所有变量的存储空间相邻吗?tc2.0中，整型、字符型、长整型数据的存储空间分别为多大?
2. 不同的数据类型对数据运算方式的有何影响?

研究过程

- 通过`tcc`生成`asm`文件

```
    ifndef  ??version
?debug  macro
    endm
    endif
    ?debug  S "\src\three3.c"
_TEXT   segment byte public 'CODE'
DGROUP  group   _DATA,_BSS
    assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT   ends
_DATA   segment word public 'DATA'
d@  label   byte
d@w label   word
_DATA   ends
_BSS    segment word public 'BSS'
b@  label   byte
b@w label   word
    ?debug  C E9496C39510D5C7372635C7468726565332E63
_BSS    ends
_DATA   segment word public 'DATA'
_a  label   word
    dw  1
_b  label   word
    dw  1
_c  label   byte
    db  1
_a1 label   word
    dw  1
_a2 label   word
    dw  1
```

```
30      dw  0
31  _DATA   ends
32  _TEXT   segment byte public 'CODE'
33  ;   ?debug  L 7
34  _main   proc    near
35  ;   ?debug  L 8
36      inc word ptr DGROUP:_a
37  ;   ?debug  L 9
38      inc word ptr DGROUP:_b
39  ;   ?debug  L 10
40      inc byte ptr DGROUP:_c
41  ;   ?debug  L 11
42      inc word ptr DGROUP:_a1
43  ;   ?debug  L 12
44      add word ptr DGROUP:_a2,1
45      adc word ptr DGROUP:_a2+2,0
46  @1:
47  ;   ?debug  L 13
48      ret
49  _main   endp
50  _TEXT   ends
51      ?debug  C E9
52  _DATA   segment word public 'DATA'
53  s@  label   byte
54  _DATA   ends
55  _TEXT   segment byte public 'CODE'
56  _TEXT   ends
57      public  _main
58      public  _c
59      public  _b
60      public  _a
61      public  _a2
62      public  _a1
63      end
64  .
```

- 通过汇编代码可以发现只有长整型的运算方式是和其余数据类型的运算方式有所不同，长整型的运算方式为带进位的加法
- 通过汇编代码可以看出，整型的存储空间为一个字，字符型的存储空间为一个字节，长整型的存储空间为两个字大小。
- 修改程序然后生成可执行文件
    - 修改后将数据改成易辨识的数据（1，2，3，4，5）

```
1   unsigned int a = 1;
2   unsigned int b = 2;
3   unsigned char c = 3;
4   unsigned int a1 = 4;
5   unsigned long a2 = 5;
6
7   main() {
8       a++;
9       b++;
10      c++;
11      a1++;
12      a2++;
13  }
```

- 通过debug来进行查看变量的存储空间是否相邻

```
-g 01fa

AX=0000  BX=0242  CX=000D  DX=C7AD  SP=FFDE  BP=FFE8  SI=003A  DI=0235
DS=07C1  ES=07C1  SS=07C1  CS=076A  IP=01FA   NV UP EI PL ZR NA PE NC
076A:01FA FF069401      INC     WORD PTR [0194]                  DS:0194=0001
-d ds
     ^ Error
-d ds:0194
07C1:0190                01 00 02 00-03 04 00 05 00 00 00 00      ............
07C1:01A0  15 02 15 02 15 02 00 00-00 10 00 00 D2 01 D2 01   ................
07C1:01B0  D9 01 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
07C1:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
07C1:01D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
07C1:01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
07C1:01F0  00 00 38 02 00 00 F8 01-41 00 50 FF 50 41 54 48   ..8.....A.P.PATH
07C1:0200  3D 5A 3A 5C 00 43 4F 4D-53 50 45 43 3D 5A 3A 5C   =Z:\.COMSPEC=Z:\
07C1:0210  43 4F 4D 4D                                       COMM
-_;
```

- 从图中可以看出存储空间依次相邻

研究4

摘要

```c
/*定义一个结构体类型stu，这个结构体类型描述一个学生的成绩。其中包括
    : 学号（整型）、C、Os、masm三门课程的成绩（字符型）*/
struct stu {
    int number;
    char c;
    char os;
    char masm;
};
/*注意，在实际程序设计中，程序员往往需要定义新的数据类型来对数据进行抽象。C语言支持用基本的数据类型如“char”、“int”等等来构造新的更为复杂的数据类型。

“整型”这一数据类型的名称为“int”:“字符型这一数据类型的名称为“char”:
以上定义了一个 新的数据类型“struct stu”，这个数据类型的名称为“stu

structstu型数据包括4个数据项 : number、c、os、masm*/

struct stu a; /*定义一个struct stu型的变量a*/

main() {
    /*定义一个struct stu型的变量b*/
    struct stu b;

    a.number = 1;
    a.c = 80;
    a.os = 82;
    a.masm = 88;

    b.number = 2;
    b.c = 90;
    b.os = 92;
    b.masm = 98;

    printf("number c  os masm\n");
```

```
33      printf("-----------------\n");
34
35      printf("%d     %d  %d  %d\n", a.number, a.c, a.os, a.masm);
36      printf("%d     %d  %d  %d\n", b.number, b.c, b.os, b.masm);
37  }
```

问题

- 变量`a`，`b`和他们的各个数据项的存储空间如何分配

研究过程

- 将程序通过`tcc`生成汇编文件

```
1       ifndef  ??version
2   ?debug  macro
3       endm
4       endif
5       ?debug  S "\src\three4.c"
6   _TEXT   segment byte public 'CODE'
7   DGROUP  group   _DATA,_BSS
8       assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
9   _TEXT   ends
10  _DATA   segment word public 'DATA'
11  d@  label   byte
12  d@w label   word
13  _DATA   ends
14  _BSS    segment word public 'BSS'
15  b@  label   byte
16  b@w label   word
17      ?debug  C E9127239510D5C7372635C7468726565342E63
18  _BSS    ends
19  _TEXT   segment byte public 'CODE'
20  ;   ?debug  L 18
21  _main   proc    near
22      push    bp
23      mov bp,sp
24      sub sp,6
25  ;   ?debug  L 22
26      mov word ptr DGROUP:_a,1
27  ;   ?debug  L 23
28      mov byte ptr DGROUP:_a+2,80
29  ;   ?debug  L 24
30      mov byte ptr DGROUP:_a+3,82
31  ;   ?debug  L 25
32      mov byte ptr DGROUP:_a+4,88
33  ;   ?debug  L 27
34      mov word ptr [bp-6],2
35  ;   ?debug  L 28
36      mov byte ptr [bp-4],90
37  ;   ?debug  L 29
38      mov byte ptr [bp-3],92
39  ;   ?debug  L 30
40      mov byte ptr [bp-2],98
41  ;   ?debug  L 32
```

```
42      mov ax,offset DGROUP:s@
43      push    ax
44      call    near ptr _printf
45      pop cx
46  ;   ?debug  L 33
47      mov ax,offset DGROUP:s@+19
48      push    ax
49      call    near ptr _printf
50      pop cx
51  ;   ?debug  L 35
52      mov al,byte ptr DGROUP:_a+4
53      cbw
54      push    ax
55      mov al,byte ptr DGROUP:_a+3
56      cbw
57      push    ax
58      mov al,byte ptr DGROUP:_a+2
59      cbw
60      push    ax
61      push    word ptr DGROUP:_a
62      mov ax,offset DGROUP:s@+38
63      push    ax
64      call    near ptr _printf
65      add sp,10
66  ;   ?debug  L 36
67      mov al,byte ptr [bp-2]
68      cbw
69      push    ax
70      mov al,byte ptr [bp-3]
71      cbw
72      push    ax
73      mov al,byte ptr [bp-4]
74      cbw
75      push    ax
76      push    word ptr [bp-6]
77      mov ax,offset DGROUP:s@+57
78      push    ax
79      call    near ptr _printf
80      add sp,10
81  @1:
82  ;   ?debug  L 37
83      mov sp,bp
84      pop bp
85      ret
86  _main   endp
87  _TEXT   ends
88  _BSS    segment word public 'BSS'
89  _a  label   word
90      db  5 dup (?)
91  _BSS    ends
92      ?debug  C E9
93  _DATA   segment word public 'DATA'
94  s@  label   byte
95      db  110
96      db  117
97      db  109
98  ..........
99      db  0
```

```
_DATA   ends
_TEXT   segment byte public 'CODE'
    extrn   _printf:near
_TEXT   ends
    public  _main
    public  _a
    end
```

- 观察汇编文件可以发现，变量 `a` ， `b` 的各个数据项的存储空间是连续分配的不过 `a` 是在数据段中而 `b` 是在栈中，而变量a和b是存储的该变量的初始地址

研究5

摘要

```c
struct n {
    int a;
    int b;
    int c;
};

int f(struct n);

struct n func(void);

main() {
    struct n a;
    int b;

    a = func();
    b = f(a);

    printf(" %d", b);
    printf(" %d", f(func()));
}

int f(struct n a) { return (a.a + a.b) * a.c; }

struct n func(void) {
    struct n a;
    a.a = 1;
    a.b = 2;
    a.c = 3;
    return a;
}
```

问题

- 向函数传递结构体型数据是如何实现的?
- 从函数返回的结构体型数据存储在何处?

研究过程

- 要探讨结构体是如何返回的，那么首先应该观察执行函数前后有什么变化。我们开始执行到调用func的汇编语句的前一句然后在执行到后一句观察寄存器的值的变化

```
076A:01FB 8BEC          MOV     BP,SP
076A:01FD 83EC06        SUB     SP,+06
076A:0200 56            PUSH    SI
076A:0201 8D5EFA        LEA     BX,[BP-06]
076A:0204 16            PUSH    SS
076A:0205 53            PUSH    BX
076A:0206 E85D00        CALL    0266
076A:0209 1E            PUSH    DS
076A:020A 50            PUSH    AX
076A:020B B90600        MOV     CX,0006
076A:020E 9AEA136A07    CALL    076A:13EA
076A:0213 8D5EFA        LEA     BX,[BP-06]
076A:0216 8CD2          MOV     DX,SS
076A:0218 8BC3          MOV     AX,BX
-g 206

AX=0000  BX=FFD6  CX=000F  DX=4464  SP=FFD0  BP=FFDC  SI=003A  DI=04B3
DS=08AD  ES=08AD  SS=08AD  CS=076A  IP=0206   NV UP EI NG NZ NA PO NC
076A:0206 E85D00        CALL    0266
-p                              发现执行完前后ax 的值改变

AX=042A  BX=FFC6  CX=0000  DX=4464  SP=FFD0  BP=FFDC  SI=003A  DI=04B3
DS=08AD  ES=08AD  SS=08AD  CS=076A  IP=0209   NV UP EI PL ZR NA PE NC
076A:0209 1E            PUSH    DS
- ; ;_
```

  - 发现ax的值发生变化然后看对应数据段中的数据，发现从函数返回的结构体型数据存储在数据段中

```
076A:0205 53            PUSH    BX
076A:0206 E85D00        CALL    0266
076A:0209 1E            PUSH    DS
076A:020A 50            PUSH    AX
076A:020B B90600        MOV     CX,0006
076A:020E 9AEA136A07    CALL    076A:13EA
076A:0213 8D5EFA        LEA     BX,[BP-06]
076A:0216 8CD2          MOV     DX,SS
076A:0218 8BC3          MOV     AX,BX
-g 209

AX=042A  BX=FFC6  CX=0000  DX=5EF6  SP=FFD0  BP=FFDC  SI=003A  DI=04B3
DS=08AD  ES=08AD  SS=08AD  CS=076A  IP=0209   NV UP EI PL ZR NA PE NC
076A:0209 1E            PUSH    DS
-d ds:042a
08AD:0420                               01 00 02 00 03 00              ......
08AD:0430  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
08AD:0440  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
08AD:0450  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
08AD:0460  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
08AD:0470  B6 04 00 00 76 04 41 00-00 00 50 41 54 48 3D 5A   ....v.A...PATH=Z
08AD:0480  3A 5C 00 43 4F 4D 53 50-45 43 3D 5A 3A 5C 43 4F   :\.COMSPEC=Z:\CO
08AD:0490  4D 4D 41 4E 44 2E 43 4F-4D 00 42 4C 41 53 54 45   MMAND.COM.BLASTE
08AD:04A0  52 3D 41 32 32 30 20 49-37 20                     R=A220 I7
```

  - 所以从函数返回的结构体型数据存储在数据段中。

- 对于第一个问题向函数传递结构体型数据是如何实现的?

  1. 首先观察 `b = f(a);` 对应的汇编代码

```
;   ?debug  L 16
    lea bx,word ptr [bp-6]
    mov dx,ss
    mov ax,bx
    mov cx,6
    call    far ptr SPUSH@
    call    near ptr _f
    add sp,6
    mov si,ax
```

2. 可以看到在执行 `call  near ptr _f` 前先执行了 `call  far ptr SPUSH@` 根据名字猜测应该是往栈里吗压入 一些数据，接下来在 `debug` 中检验猜测是否正确
3. 观察到执行完 `call far ptr SPUSH@` 前后寄存器 `sp` 的值发生了改变然后观察栈中的数据可以看到把结构体中的数据压入到栈中(`ss:ffce`)。然后接下来进入 `_f` 中看是否使用了栈中压入的数据

```
-g 21d

AX=FFD6  BX=FFD6  CX=0006  DX=08AD  SP=FFD4  BP=FFDC  SI=003A  DI=04B3
DS=08AD  ES=08AD  SS=08AD  CS=076A  IP=021D   NV UP EI PL ZR NA PE NC
076A:021D 9A06146A07    CALL    076A:1406
-p

AX=08AD  BX=0222  CX=0000  DX=076A  SP=FFCE  BP=FFDC  SI=003A  DI=04B3
DS=08AD  ES=08AD  SS=08AD  CS=076A  IP=0222   NV UP EI PL ZR NA PE NC
076A:0222 E83100        CALL    0256
-d ss:ffce
08AD:FFC0                                               01 00           ..
08AD:FFD0  02 00 03 00 3A 00 01 00-02 00 03 00 E8 FF 1D 01   ....:...........
08AD:FFE0  01 00 E6 FF BA 04 EA FF-00 00 43 3A 5C 53 52 43   ..........C:\SRC
08AD:FFF0  5C 54 48 52 45 45 35 2E-45 58 45 00 F8 00 FB 00   \THREE5.EXE.....
- ; ; ; ; ;_
```

4. 在进入后执行到 `mov bp,sp` 后可以发现确实使用了栈中压入的数据（ `ss:ffce` ）

```
AX=08AD  BX=0222  CX=0000  DX=076A  SP=FFCA  BP=FFDC  SI=003A  DI=04B3
DS=08AD  ES=08AD  SS=08AD  CS=076A  IP=0257   NV UP EI PL ZR NA PE NC
076A:0257 8BEC          MOV     BP,SP
-t

AX=08AD  BX=0222  CX=0000  DX=076A  SP=FFCA  BP=FFCA  SI=003A  DI=04B3
DS=08AD  ES=08AD  SS=08AD  CS=076A  IP=0259   NV UP EI PL ZR NA PE NC
076A:0259 8B4604        MOV     AX,[BP+04]                      SS:FFCE=0001
- ;_
```

5. 到此就可以回答了，函数是通过栈来传递结构体的

总结

- 变量无非就是存储在栈中或者数据段中，如果变量需要长期存在或者要返回那么会将变量存储在数据段中，而如过仅仅只需在函数中存在或者传递参数就需要将数据保存在栈中。