> 2022.10.20
>
> 笔记 by 刘元

调用约定

- `__cdecl`
- `__stdcall`
- `__fastcall`

内联汇编与裸函数

- 内联汇编
- 裸函数

调用约定

> 需要注意的是，我们讨论的调用约定谈论的是C/C++，再高级的语言都用虚拟机做了处理

关于调用约定，需要考虑的问题有二：

- 传参的顺序？意即，是从左向右传参，还是从右向左传参？

  在本篇笔记中，三种方式均为从右向左传参。
- 如何平衡堆栈？意即，传参产生的额外空间占用，如何归还？

平栈分为内平栈与外平栈。考虑如下C代码：

```c
int test(int a, int b)
{
    return a + b;
}

int main()
{
    test(1, 2);
}
```

假如我们在调用 `test` 之后，由 `main` 来对栈进行平衡，就称为外平栈：

```
1  00DB13DE  push        2
2  00DB13E0  push        1
3  00DB13E2  call        _test (0DB11E0h)
4  00DB13E7  add         esp,8  ; 在调用结束后由调用函数进行平栈
```

假如我们在调用的`test`内进行平栈，就称为内平栈：

```
1  ; 恢复现场
2  00D21434  pop         edi
3  00D21435  pop         esi
4  00D21436  pop         ebx
5  ; 恢复栈帧
6  00D21437  mov         esp,ebp
7  00D21439  pop         ebp
8  ; 平栈
9  00D2143A  ret         8
```

`__cdecl`

> Windows默认方式

```
1  00B713DE  push        2
2  00B713E0  push        1
3  00B713E2  call        _test (0B711EAh)
4  00B713E7  add         esp,8
```

1. 传参方式：从右向左传参
2. 平栈方式：外平栈

`__stdcall`

> Windows的动态链接库默认的方式

`main`内：

```
1  007413DE  push        2
2  007413E0  push        1
3  007413E2  call        _test@8 (07411EFh)
```

`test` 内：

```
1  0074142E  mov         eax,dword ptr [a]
2  00741431  add         eax,dword ptr [b]
3
4  00741434  pop         edi
5  00741435  pop         esi
6  00741436  pop         ebx
7  00741437  mov         esp,ebp
8  00741439  pop         ebp
9  0074143A  ret         8
```

1. 传参方式：从右向左
2. 平栈方式：内平栈

`__fastcall`

> Linux默认的传参方式

1. 传参方式：纯靠寄存器传参，寄存器不够再用栈传参
   - 32位：ecx和edx

```
1      test(1, 2, 3);
2  009C141E  push        3
3  009C1420  mov         edx,2
4  009C1425  mov         ecx,1
5  009C142A  call        @test@12 (09C11F4h)
```

   - 64位：edi，esi，edx，ecx，r8d，r9d
2. 平栈方式：
   - Linux：外平栈

     在本例中，我们的 `test` 接收7个参数：

```
0x00000000004004fc <+4>:     push   0x7
0x00000000004004fe <+6>:     mov    r9d,0x6
0x0000000000400504 <+12>:    mov    r8d,0x5
0x000000000040050a <+18>:    mov    ecx,0x4
0x000000000040050f <+23>:    mov    edx,0x3
0x0000000000400514 <+28>:    mov    esi,0x2
0x0000000000400519 <+33>:    mov    edi,0x1
0x000000000040051e <+38>:    call   0x4004d6 <test>
0x0000000000400523 <+43>:    add    rsp,0x8
```

- Windows：内平栈

```
010013EC  pop         edi
010013ED  pop         esi
010013EE  pop         ebx
010013EF  mov         esp,ebp
010013F1  pop         ebp
010013F2  ret         4
```

内联汇编与裸函数

内联汇编

```
__asm__(汇编语句模板 : 输出部分 : 输入部分 : 破坏描述部分)
```

https://www.cnblogs.com/kuangke/p/12456943.html

裸函数

> Windows下嵌入汇编的方式，只支持32位

```
#include <stdio.h>
#include <stdlib.h>
```

```
3
4  // 普通函数
5  int add(int a, int b)
6  {
7      return a + b;
8  }
9
10 // 裸函数，编写的汇编是什么样子，生成的机器码就是汇编直接对应的机器码
11 // 换言之，编译器不会做额外处理
12 int __declspec(naked) add_naked(int a, int b)
13 {
14     __asm
15     {
16         mov eax, dword ptr ds : [esp + 8];
17         add eax, dword ptr ds : [esp + 4];
18         ret
19     }
20 }
21
22 // 普通的内联汇编
23 int add_asm(int a, int b)
24 {
25     __asm
26     {
27         mov eax, a;
28         add eax, b;
29         ret
30     }
31 }
32
33 int main()
34 {
35     int r1 = add(1, 2);
36     int r2 = add_naked(1, 2);
37     int r3 = add_asm(1, 2);
38
39     printf("r1 = %d, r2 = %d, r3 = %d\n", r1, r2, r3);
40
41     system("pause");
42     return 0;
43 }
```

反汇编：

1. 裸函数：

```
    __asm
    {
        mov eax, dword ptr ds : [esp + 12];
00371000  mov         eax,dword ptr ds:[esp+0Ch]
        add eax, dword ptr ds : [esp + 8];
00371005  add         eax,dword ptr ds:[esp+8]
        add eax, dword ptr ds : [esp + 4];
0037100A  add         eax,dword ptr ds:[esp+4]
        ret
0037100F  ret
    }
}
```

2. 内联汇编

```
int __fastcall add_asm(int a, int b, int c)
{
00371010  push        ebp
00371011  mov         ebp,esp
00371013  sub         esp,8
00371016  mov         dword ptr [ebp-8],edx
00371019  mov         dword ptr [ebp-4],ecx
    __asm
    {
        mov eax, a;
0037101C  mov         eax,dword ptr [ebp-4]
        add eax, b;
0037101F  add         eax,dword ptr [ebp-8]
        add eax, c;
00371022  add         eax,dword ptr [c]
        ret
00371025  ret
    }
}
00371026  mov         esp,ebp
00371028  pop         ebp
00371029  ret         4
```

3. `main` 函数：

```
int r1 = add(1, 2, 3);
    int r2 = add_naked(1, 2, 3);
```

```
 3  00371030  mov         edx,2
 4  00371035  push        esi
 5  00371036  push        3
 6  00371038  lea         ecx,[edx-1]
 7  0037103B  call        add_naked (0371000h)
 8      int r3 = add_asm(1, 2, 3);
 9  00371040  push        3
10  00371042  mov         esi,eax
11  00371044  call        add_asm (0371010h)
12
13      printf("r1 = %d, r2 = %d, r3 = %d\n", r1, r2, r3);
14  00371049  push        eax
15  0037104A  push        esi
16  0037104B  push        3
17  0037104D  push        372100h
18  00371052  call        dword ptr ds:[372090h]
19
20      system("pause");
21  00371058  push        37211Ch
22  0037105D  call        dword ptr ds:[372094h]
23  00371063  add         esp,14h
24      return 0;
25  00371066  xor         eax,eax
26  00371068  pop         esi
27  }
28  00371069  ret
```

4. 完整的反汇编代码：

```
 1  --- e:\desktop\project1\project1\main.c ---------------------------------------
 2      __asm
 3      {
 4          mov eax, dword ptr ds : [esp + 12];
 5  00371000  mov         eax,dword ptr ds:[esp+0Ch]
 6          add eax, dword ptr ds : [esp + 8];
 7  00371005  add         eax,dword ptr ds:[esp+8]
 8          add eax, dword ptr ds : [esp + 4];
 9  0037100A  add         eax,dword ptr ds:[esp+4]
10          ret
11  0037100F  ret
12      }
13  }
14
15  int __fastcall add_asm(int a, int b, int c)
16  {
```

```
17  00371010  push        ebp
18  00371011  mov         ebp,esp
19  00371013  sub         esp,8
20  00371016  mov         dword ptr [ebp-8],edx
21  00371019  mov         dword ptr [ebp-4],ecx
22      __asm
23      {
24          mov eax, a;
25  0037101C  mov         eax,dword ptr [ebp-4]
26          add eax, b;
27  0037101F  add         eax,dword ptr [ebp-8]
28          add eax, c;
29  00371022  add         eax,dword ptr [c]
30          ret
31  00371025  ret
32      }
33  }
34  00371026  mov         esp,ebp
35  00371028  pop         ebp
36  00371029  ret         4
37  --- 无源文件 ------------------------------------------------------------------
    -
38  0037102C  int         3
39  0037102D  int         3
40  0037102E  int         3
41  0037102F  int         3
42  --- e:\desktop\project1\project1\main.c ----------------------------------------
43      int r1 = add(1, 2, 3);
44      int r2 = add_naked(1, 2, 3);
45  00371030  mov         edx,2
46  00371035  push        esi
47  00371036  push        3
48  00371038  lea         ecx,[edx-1]
49  0037103B  call        add_naked (0371000h)
50      int r3 = add_asm(1, 2, 3);
51  00371040  push        3
52  00371042  mov         esi,eax
53  00371044  call        add_asm (0371010h)
54
55      printf("r1 = %d, r2 = %d, r3 = %d\n", r1, r2, r3);
56  00371049  push        eax
57  0037104A  push        esi
58  0037104B  push        3
59  0037104D  push        372100h
60  00371052  call        dword ptr ds:[372090h]
61
```

```
62      system("pause");
63  00371058  push        37211Ch
64  0037105D  call        dword ptr ds:[372094h]
65  00371063  add         esp,14h
66      return 0;
67  00371066  xor         eax,eax
68  00371068  pop         esi
69  }
70  00371069  ret
```