함수를 호출하는 형태에는 여러 가지 방법이 있다. Visual C++에서는 네 가지 형태의 호출 규약을 지원한다. \_\_cdecl, \_\_fastcall, \_\_stdcall, \_\_thiscall이 그것이다. 각 호출 규약은 비슷하지만 약간 씩 차이를 가지고 있다. 각각의 호출 규약에 대한 특징을 살펴 보도록 하자.

### \_\_cdecl

C언어 표준 호출 규약이다. 파라미터는 오른쪽에서 왼쪽으로 스택을 통해 전달되며, 호출한 곳에서 스택을 정리한다. 특징적인건 호출한 쪽에서 스택을 정리하기 때문에 가변 인자를 지원한다는 것이다.

1. extern "C" int \_\_cdecl CdeclFunc(int a, int b, int c)
2. {
3. printf("%d %d %d\n", a, b, c);
4. }

위 함수를 호출하기 위한 어셈블리 코드는 아래와 같다. add 명령어를 통해서 호출 한 후에 스택을 정리해 주어야 한다.

push 3

push 2

push 1

call CdeclFunc

add esp, 12

**\_\_fastcall**

말 그대로 빠른 호출이다. 파라미터 중 일부를 레지스터를 통해서 전달하는 함수다. x86 계열에서는 일반적으로 ecx, edx로 파라미터를 전달하고 나머지는 스택으로 전달한다. \_\_cdecl과 같이 오른쪽에서 왼쪽으로 파라미터를 전달하며, 스택 정리는 호출을 당한 곳에서 수행한다. 따라서 가변 인자를 지원하지 못한다.

1. extern "C" int \_\_fastcall FastcallFunc(int a, int b, int c)
2. {
3. printf("%d %d %d\n", a, b, c);
4. }

위 함수를 호출 하는 코드는 아래와 같다. 앞 쪽 두 개의 파라미터가 ecx와 edx를 통해서 전달된다. 파라미터가 두 개 이하인 함수에 대해서는 스택을 전혀 사용하지 않기 때문에 빠르게 동작한다.

push 3

mov edx, 2

mov ecx, 1

call @FastcallFunc@12

**\_\_stdcall**

윈도우 API의 표준 호출 규약이다. 파라미터는 오른쪽에서 왼쪽으로 스택을 통해서 전달되며, 스택 정리는 호출 당한 곳에서 이루어진다.

1. extern "C" int \_\_stdcall StdcallFunc(int a, int b, int c)
2. {
3. printf("%d %d %d\n", a, b, c);
4. }

호출 코드는 아래와 같다. \_\_cdecl과 유사하지만 스택 정리 코드가 없는게 특징이다.

push 3

push 2

push 1

call \_StdcallFunc@12

**\_\_thiscall**

직접적으로 함수 호출 규약으로 사용할 수는 없다. 이 호출 규약은 C++의 클래스 멤버 함수 호출 규약으로 사용된다. 파라미터는 오른쪽에서 왼쪽으로 스택을 통해서 전달되고, 호출 당한 곳에서 스택을 정리한다. 특징적 인건 ecx를 통해서 클래스 포인터를 전달하는 점이다.

1. class CCallConv
2. {
3. public:
4. int ThisCall(int a, int b, int c);
5. };
6.
7. int CCallConv::ThisCall(int a, int b, int c)
8. {
9. return printf("%d %d %d\n", a, b, c);
10. }

호출 하는 코드는 아래와 같다. 특징은 ecx로 클래스의 this 포인터를 전달한다는 점이다.

push 3

push 2

push 1

lea ecx, conv

call CCallConv::ThisCall

**스택 정리 방식의 차이**

함수 호출 규약의 주된 차이점은 스택 정리 방식이다. 호출한 쪽에서 정리하는 방식과 호출을 당한 곳에서 정리하는 방법이 있다. 호출한 쪽에서 정리하는 방법의 장점은 자신이 파라미터로 전달한 인자의 개수를 정확히 알 수 있기 때문에 가변 인자를 지원할 수 있다는 것이다. 그렇다면 \_\_cdecl을 제외한 나머지 방식은 왜 스택을 호출을 당한 곳에서 정리하는 것일까?

그 이유는 속도에 있다. x86 CPU의 경우 ret 어셈블리 명령어를 두 가지 형태로 지원한다. ret를 하면 단순히 스택에 저장된 복귀 주소로 리턴 한다. 하지만 ret imm16을 사용하면 imm16만큼 스택에서 팝 한 다음 꺼내진 복귀 주소로 리턴 한다. 위에서 살펴 보았듯이 \_\_cdecl은 스택 정리를 위해서 add 명령어가 추가된다. 반면에 호출을 당한 곳에서 정리하는 방식은 리턴할 때에 ret imm16을 사용함으로써 add 명령어가 추가되지 않아도 된다. 또한 486을 기준으로 했을 때 ret와 ret imm16은 5클럭 사이클을 소모하는 동일한 명령어로 처리된다. 따라서 \_\_cdecl보다는 다른 호출 규약이 근소하게 빠를 수 있다.

**멤버 함수는 모두 \_\_thiscall??**

위에서 멤버 함수엔 \_\_thiscall을 사용한다고 했다. 물론 기본적으로 \_\_thiscall이 사용되나, 직접 지정할 경우 다른 호출 규약을 사용할 수 있다. 다른 호출 규약을 지정하게 되면 첫 번째 인자로 this 포인터가 전달된다.

**정리**

|  |  |  |  |
| --- | --- | --- | --- |
| 호출 규약 | 파라미터 | 스택 | 특징 |
| \_\_cdecl | 오른쪽 -> 왼쪽 | 호출한 곳 | 가변 인자 지원 |
| \_\_fastcall | 오른쪽 -> 왼쪽 | 호출 당한 곳 | 파라미터 두 개를 ecx, edx를 통해서 전달하기 때문에 두 개 이하의 인자를 가진 함수에 대해서 빠름 |
| \_\_stdcall | 오른쪽 -> 왼쪽 | 호출 당한 곳 | Windows 표준 호출 규약 |
| \_\_thiscall | 오른쪽 -> 왼쪽 | 호출 당한 곳 | ecx를 통해서 this 포인터를 전달 함 |

**코드**

테스트에 사용된 C언어 코드다.

1. extern "C" int \_\_cdecl CdeclFunc(int a, int b, int c)
2. {
3. return printf("%d %d %d\n", a, b, c);
4. }
5.
6.
7. extern "C" int \_\_fastcall FastcallFunc(int a, int b, int c)
8. {
9. return printf("%d %d %d\n", a, b, c);
10. }
11.
12.
13. extern "C" int \_\_stdcall StdcallFunc(int a, int b, int c)
14. {
15. return printf("%d %d %d\n", a, b, c);
16. }
17.
18. class CCallConv
19. {
20. public:
21. int ThisCall(int a, int b, int c);
22. };
23.
24. int CCallConv::ThisCall(int a, int b, int c)
25. {
26. return printf("%d %d %d\n", a, b, c);
27. }
28.
29. int \_tmain(int argc, \_TCHAR\* argv[])
30. {
31. CdeclFunc(1, 2, 3);
32. FastcallFunc(1, 2, 3);
33. StdcallFunc(1, 2, 3);
34.
35. CCallConv conv;
36. conv.ThisCall(1,2,3);
37. return 0;
38. }

CdeclFunc 함수의 어셈블리 리스팅이다.

PUBLIC \_CdeclFunc

EXTRN \_printf:NEAR

; Function compile flags: /Odt

\_TEXT SEGMENT

\_a$ = 8 ; size = 4

\_b$ = 12 ; size = 4

\_c$ = 16 ; size = 4

\_CdeclFunc PROC NEAR

; File d:\test\realtest\callconv\callconv.cpp

; Line 7

 push ebp

 mov ebp, esp

; Line 8

 mov eax, DWORD PTR \_c$[ebp]

 push eax

 mov ecx, DWORD PTR \_b$[ebp]

 push ecx

 mov edx, DWORD PTR \_a$[ebp]

 push edx

 push OFFSET FLAT:$SG9623

 call \_printf

 add esp, 16 ; 00000010H

; Line 9

 pop ebp

 ret 0

\_CdeclFunc ENDP

FastcallFunc 함수의 어셈블리 리스팅이다.

\_TEXT ENDS

PUBLIC @FastcallFunc@12

; Function compile flags: /Odt

\_TEXT SEGMENT

\_b$ = -8 ; size = 4

\_a$ = -4 ; size = 4

\_c$ = 8 ; size = 4

@FastcallFunc@12 PROC NEAR

; \_a$ = ecx

; \_b$ = edx

; Line 13

 push ebp

 mov ebp, esp

 sub esp, 8

 mov DWORD PTR \_b$[ebp], edx

 mov DWORD PTR \_a$[ebp], ecx

; Line 14

 mov eax, DWORD PTR \_c$[ebp]

 push eax

 mov ecx, DWORD PTR \_b$[ebp]

 push ecx

 mov edx, DWORD PTR \_a$[ebp]

 push edx

 push OFFSET FLAT:$SG9629

 call \_printf

 add esp, 16 ; 00000010H

; Line 15

 mov esp, ebp

 pop ebp

 ret 4

@FastcallFunc@12 ENDP

\_TEXT ENDS

StdcallFunc 함수의 어셈블리 리스팅이다.

PUBLIC \_StdcallFunc@12

; Function compile flags: /Odt

\_TEXT SEGMENT

\_a$ = 8 ; size = 4

\_b$ = 12 ; size = 4

\_c$ = 16 ; size = 4

\_StdcallFunc@12 PROC NEAR

; Line 19

 push ebp

 mov ebp, esp

; Line 20

 mov eax, DWORD PTR \_c$[ebp]

 push eax

 mov ecx, DWORD PTR \_b$[ebp]

 push ecx

 mov edx, DWORD PTR \_a$[ebp]

 push edx

 push OFFSET FLAT:$SG9635

 call \_printf

 add esp, 16 ; 00000010H

; Line 21

 pop ebp

 ret 12 ; 0000000cH

\_StdcallFunc@12 ENDP

\_TEXT ENDS

ThisCall 함수의 어셈블리 리스팅이다.

PUBLIC ?ThisCall@CCallConv@@QAEHHHH@Z ; CCallConv::ThisCall

; Function compile flags: /Odt

\_TEXT SEGMENT

\_this$ = -4 ; size = 4

\_a$ = 8 ; size = 4

\_b$ = 12 ; size = 4

\_c$ = 16 ; size = 4

?ThisCall@CCallConv@@QAEHHHH@Z PROC NEAR ; CCallConv::ThisCall

; \_this$ = ecx

; Line 30

 push ebp

 mov ebp, esp

 push ecx

 mov DWORD PTR \_this$[ebp], ecx

; Line 31

 mov eax, DWORD PTR \_c$[ebp]

 push eax

 mov ecx, DWORD PTR \_b$[ebp]

 push ecx

 mov edx, DWORD PTR \_a$[ebp]

 push edx

 push OFFSET FLAT:$SG9653

 call \_printf

 add esp, 16 ; 00000010H

; Line 32

 mov esp, ebp

 pop ebp

 ret 12 ; 0000000cH

?ThisCall@CCallConv@@QAEHHHH@Z ENDP ; CCallConv::ThisCall

\_TEXT ENDS

main 함수의 어셈블리 리스팅이다.

PUBLIC \_main

; Function compile flags: /Odt

\_TEXT SEGMENT

\_conv$ = -1 ; size = 1

\_argc$ = 8 ; size = 4

\_argv$ = 12 ; size = 4

\_main PROC NEAR

; Line 35

 push ebp

 mov ebp, esp

 push ecx

; Line 36

 push 3

 push 2

 push 1

 call \_CdeclFunc

 add esp, 12 ; 0000000cH

; Line 37

 push 3

 mov edx, 2

 mov ecx, 1

 call @FastcallFunc@12

; Line 38

 push 3

 push 2

 push 1

 call \_StdcallFunc@12

; Line 41

 push 3

 push 2

 push 1

 lea ecx, DWORD PTR \_conv$[ebp]

 call ?ThisCall@CCallConv@@QAEHHHH@Z ; CCallConv::ThisCall

; Line 42

 xor eax, eax

; Line 43

 mov esp, ebp

 pop ebp

 ret 0

\_main ENDP

**참고 자료**

<http://www.unixwiz.net/techtips/win32-callconv-asm.html>
<http://home.comcast.net/~fbui/intel.html#clock>
<http://www.hardwaresecrets.com/article/270/4>
<http://swox.com/doc/x86-timing.pdf>
<http://developer.intel.com/design/Pentium4/documentation.htm>