1.함수의 호출

함수를 호출항때 어떻게 스택에 PUSH하고 POP할까?

함수호출은 크게 다음과 같이 나뉘어 진다.

1. 함수가 사용할 파라미터를 스택에 넣고 함수 시작지점으로 점프(함수호출)을 한다.
2. 함수 내에서 사용할 스택 프레임을 설정한다. (프롤로그)
3. 함수의 내용을 수행한다.
4. 수행을 마치고 처음 호출한 지점으로 돌아가기 위해 스택을 복원한다. (에필로그)

호출한 지점의 다음으로 점프한다.

2번 과정을 프롤로그(prolog) 라고 부르며, 4번 과정을 에필로그(epilog)라고 한다.

2.함수 프롤로그&에필로그

함수의 포롤로그와 에필로그는 함수호출 규약에 따라 조금씩 다르다.

대표적으로 cdecl을 보면 다음과 같다.

cdecl : C 프로그래밍 언어가 기원인 호출 규약으로서 x86 아키텍처용의 수많은 C 컴파일러가 사용한다.

cdecl

특징

1. 스택에 파라메터 삽입 순서 : right → left
2. 스택의 정리를 호출한 함수(caller)에서 수행한다. 따라서 가변인자를 사용할 수 있다.
3. Name Mangling : 함수 이름 앞에 _ 추가
   ex) _Foo
4. C / C++ 언어의 기본 함수 호출규약

```
int main() {
    foo();
    /* foo 함수 호출
        call foo (128102Dh)
        add esp, 8 ;        (6) <<스택을 정리하는 부분>>
        */
    return 0;
}
```

위 코드를 보면 역순으로 파라메터를 push 하고 함수호출을 한 후, 이어지는 다음 라인에서 스택을 정리함을 알 수 있다. 즉 caller 가 직접 스택을 정리한다.

함수 프롤로그

```
/* caller 의 ebp 를 저장하고 callee (foo) 의 ebp 를 확보
    push ebp
    mov ebp, esp
    push ecx ; local 변수 c 의 자리 확보
*/
```

베이스포인터(ebp)를 스택에 저장하고 현재 스택포인터(esp)를 베이스포인터(ebp)에다가 저장함.

함수 에필로그

```
/* foo 함수 종료 후 caller 의 ebp, esp 복구
    mov esp, ebp
    pop ebp
    ret
*/
```

현재 스택 포인터(esp)를 베이스포인터(ebp)로 복구한 후 베이스 포인터(ebp)를 복구해주고 ret를 통해 다음에 가야에 adress로 점프한다.

3.스텍 프레임

4.함수호출 규약

- 함수 호출 규약은 아래 4가지의 값을 기준으로 그 종류가 나뉘어진다

1. Parameter 전달 방법

   : 스택 프레임 사용해서 파라메터 전달, 레지스터 사용해서 파라메터 전달

2. Parameter 전달 순서

: 함수명(파라메터1, 파라메터2, 파라메터3, ...) 에서 어떤 파라메터부터 먼저 전달 할 것인가?

3. 함수 리턴 값 전달 방법

   : 함수 리턴 값을 어디에 저장해서, 돌려줄 것인가?

4. 함수 호출간 사용했던 stack frame을 정리하는 방법

   : 함수 사용이 끝난후에, 사용했던 stack frame을 공간을 누가 정리할 것인가?

-종류

: _cdecl, _stdcall, _fastcall 크게 3가지 있다.

1. _cdecl (C언어에서 만든 함수 호출 규약)

Parameter 전달 방법	**스택 프레임 사용해서 파라메터 전달 (파라메터 개수 제한 X)**
Parameter 전달 순서	오른쪽 파라메터 --> 왼쪽 파라메터 ex) sum(1,2,3) 함수일 경우. 3, 2, 1 순서로 스택에 push 됨.
함수 리턴 값 전달 방법	리턴 값이 4byte 이하인 경우 - eax에 리턴값 저장 리턴 값이 8byte 이하인 경우 - 리턴값의 상위 4바이트는 edx에 저장 - 리턴값의 하위 4바이트는 eax에 저장 **※ 리턴 값이 8byte 보다 큰 경우는 X.**
stack frame 정리 방법	**caller(호출한 함수) 가 callee(호출된 함수)의 stack frame 공간을 정리함** **# stack frame 정리하는 어셈 코드** **add esp, (total parameter size)**
Name Decoration	**func --> _func**

2. _stdcall (윈도우에서 사용하고 있는 표준 호출규약.)

windows API는 stdcall을 따른다.

Parameter 전달 방법	**스택 프레임 사용해서 파라메터 전달 (파라메터 개수 제한 X)**
Parameter 전달 순서	오른쪽 파라메터 --> 왼쪽 파라메터 ex) sum(1,2,3) 함수일 경우. 3, 2, 1 순서로 스택에 push 됨.
함수 리턴 값 전달 방법	리턴 값이 4byte 이하인 경우 - eax에 리턴값 저장 리턴 값이 8byte 이하인 경우 - 리턴값의 상위 4바이트는 edx에 저장 - 리턴값의 하위 4바이트는 eax에 저장 ※ 리턴 값이 8byte 보다 큰 경우는 X.
stack frame 정리 방법	**callee (호출된 함수)가 자신의 stack frame 공간을 직접 정리** **# stack frame 정리하는 어셈 코드** **보통 ret 커맨드 사용**
Name Decoration	func --> _func@파라메터 총 합

3. __fastcall (critical한 성능을 요구하는 일부 함수에서 사용. 성능의 극대화를 필요로 할 때 사용)

(윈도우 커널의 일부 함수에서 보이기는 함)

Parameter 전달 방법	**레지스터 사용해서 파라메터 전달** **- 처음 2개의 파라메터는 스택을 사용하지 않고,** **ecx와 edx 레지스터를 사용해서 전달.**
Parameter 전달 순서	오른쪽 파라메터 --> 왼쪽 파라메터 ex) sum(1,2,3) 함수일 경우. 3, 2, 1 순서로 전달 됨.
함수 리턴 값 전달 방법	리턴 값이 4byte 이하인 경우 - eax에 리턴값 저장 리턴 값이 8byte 이하인 경우 - 리턴값의 상위 4바이트는 ebx에 저장 - 리턴값의 하위 4바이트는 eax에 저장 ※ 리턴 값이 8byte 보다 큰 경우는 X.
stack frame 정리 방법	**callee(호출된 함수가 자신의 공간을 직접 정리함)**

Calling Convention	파라메터 전달 방향	파라메터 전달 방법	스택프레임 정리	Name Decoration
cdecl	R → L	Stack	Caller	_func
stdcall	R → L	Stack	Callee	_func@파라메터 총 합
fastcall	R → L	ECX, EDX, Stack	Callee	@func@파라메터 총 합