기술문서	'08. 10. 28. 작성

SEH(Structured Exception Handling)

작성자 : 영남대학교 @Xpert 정미연

e-mail : zxzxzx36@ynu.ac.kr

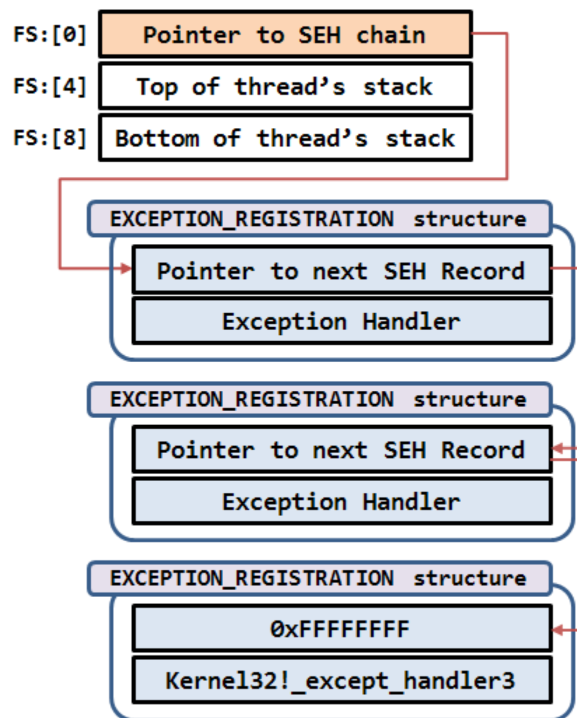
- Contents –

1. SEH란? 2
2. SEH 구조..... 2
   2-1. SEH 구조체..... 3
   2-2. Exception Handler 구조체..... 3
3. SEH 처리과정..... 5
4. SEH 생성 및 삭제..... 5
5. 참고문헌..... 8

1. SEH란?

SEH란 Structured Exception Handler의 약자로 Windows 운영체제에서 예외(Exception)를 처리하는 매커니즘이다. C++에서는 이런 Handler를 __try/__except나 __try/__finally로 선언한다. 여기서 예외(Exception)이란 프로그램이 실행될 때 일어날 수 있는 zero divide, Access violation, integer overflow 등과 같은 예외 사항을 말한다.

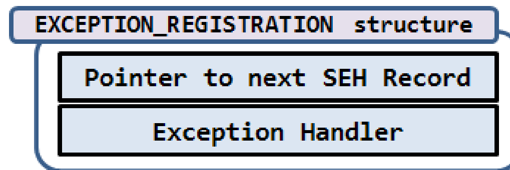
2. SEH 구조



실행 중인 스레드에 0으로 나누거나 잘못된 메모리에 접근하는 등의 예외 상황이 발생하게 되면 Windows에서는 FS 레지스터에 저장되어 있는 EXCEPTION_REGISTRATION_RECORD 값을 찾아 Exception Handler를 호출한다. 이때 해당 스레드의 EXCEPTION_REGISTRATION_RECORD 포인터는 FS:[0]에 저장이 되어있다. SEH Chain의 구조는 위의 그림과 같다.

참고로 FS:[4]에는 스레드 스택의 최상위 부분의 주소 값이 들어있고, FS:[8]에는 스레드 스택의 최하위 부분의 주소 값이 들어있다.

2-1. SEH 구조체




```
Typedef struct _EXCEPTION_REGISTRATION {
        EXCEPTION_REGISTRATION *prev;
        EXCP_HANDLER handler;
} EXCEPTION_REGISTRATION, *PEXCEPTION_REGISTRATION;
```

위의 구조체는 SEH Chain을 구성하는 구조체의 프로토타입이다.

첫 번째 파라미터는 이전에 설치되어있던 EXCEPTION_REGISTRATION_RECORD의 포인터가 저장이 되어있고, 두 번째 파라미터에는 Exception Handler의 주소 값이 저장되어있다.
SEH Chain은 첫 번째 파라미터의 값을 따라 Exception Handler를 거슬러 올라가며 예외 처리를 수행한다. 그러다가 첫 번째 파라미터 값이 0xFFFFFFFF가 되면 kernel32!_except_handler가 호출되면서 커널 영역의 예외처리 루틴을 수행하게 된다.

2-2. Exception Handler 구조체

아래의 구조체는 Exception Handler의 프로토타입이다.

```
EXCEPTION_DISPOSITION __cdecl _except_handler (
        struct _EXCEPTION_RECORD *ExceptionRecord,
        void *EstablisherFrame,
        struct _CONTEXT *ContextRecord,
        void *DispatcherContext     );
```

여기서 리턴 값인 EXCEPTION_DISPOSITION은 아래와 같다.
만약 Exception Handler 루틴에서 ExceptionContinueExecution 값을 리턴하게 되면, Exception handler는 에러가 발생했던 주소를 다시 실행시킨다. 그리고 ExceptionContinueSearch 값일 경우 Chain의 상위 Exception handler로 넘어가게 된다.

```
Typedef enum _EXCEPTION_DISPOSITION{
    ExceptionContinueExecution,    //0
    ExceptionContinueSearch,       //1
    ExceptionNestedException,      //2
    ExceptionCollidedUnwind        //3
} EXCEPTION_DISPOSITION;
```

첫 번째 파라미터 값인 EXCEPTION_RECORD의 경우는 아래와 같은 구조체로 구성이 되어있다.
이 구조체의 Exceptioncode에는 발생한 예외 종류를 나타내는 코드 값이, ExceptionFlags에는 예외상황의 플래그가 저장되어있는데 이 값이 0일 경우는 예외상황이라는 의미가 된다.
그리고 ExceptionRecord는 다른 처리되지 않은 예외상황을 위한 Exception_Record 구조의 포인터의 값이 저장되어 있다. ExceptionAddress는 예외 상황을 발생시킨 CPU 명령의 주소를 알려주고, NumberParameters는 예외 상황과 연관된 파라미터의 갯수를, ExceptionInformation는 예외 상황을 명시하는 추가적인 인자의 배열을 가지고 있다.

```
Typedef struct _EXCEPTION_RECORD {
        DWORD Exceptioncode;
        DWORD ExceptionFlags;
        Struct _ExCEPTION_RECORD *ExceptionRecord;
        PVOID ExceptionAddress;
        DWORD NumberParameters;
        UNIT_PTR ExceptionInformation [EXCEPTION_MAXIMUM_PARAMETERS];
}   EXCEPTION_RECORD;
```

두 번째 파라미터 값인 EstablisherFrame은 SEH를 설치할 때 코드에서 설치한 ESP 값을 저장하고 있다. 이때 이 값은 Chain의 상위 Exception handler를 가르키는 Pointer와 같다.
세 번째 파라미터 값인 ContextRecord는 예외가 발생할 당시의 레지스터 값들이 저장되어 있다. 이 때 사용자가 조작 불가능한 레지스터 값까지 조작할 수 있기 때문에 취약하다고 볼 수 있다.

3. SEH 처리과정

실행 중인 스레드에 0으로 나누거나 잘못된 메모리에 접근하는 등의 예외 상황이 발생하게 되면 인터럽트가 발생하게 된다. 이때 인터럽트 루틴을 실행하게 되면, Windows의 인터럽트 코드에서는 예외 발생 당시의 레지스터 값들을 스택에 저장하여 놓는다. 이때 이 값들을 트랩 프레임(Trap Frame)이라고 한다.

운영체제는 트랩 프레임을 작성한 후 운영체제 내부 함수인 KiDispatchException 함수가 호출되어 진다. 이 함수에서는 예외가 발생한 스레드를 멈춘 후 유저레벨 디버거에 LPC(local Procedure Call)와 디버그 포트를 이용하여 프로그램 예외가 발생하였음을 LPC 메시지를 통하여 전달하게 된다. LPC에 대한 메시지 전달이 완료된 후 Windows는 다시 예외가 발생한 스레드의 스택에 트랩 프레임을 써준 후 유저레벨의 ntdll에 있는 KiUserExceptionDispatcher 함수가 호출될 수 있도록 실행 흐름을 바꾼다.

4. SEH 생성 및 삭제

```
#include<windows.h>
ULONG G_nValid;
EXCEPTION_DISPOSITION __cdecl except_handler (
        struct _EXCEPTION_RECORD *ExceptionRecord, void *EstablisherFrame,
        struct _CONTEXT *ContextRecord, void *DispatcherContext) {
        void *ptr = EstablisherFrame;
        ContextRecord->Eax = (ULONG)&G_nValid;
        return ExceptionContinueExecution;
}
int main(int argc, char *argv[]) {
        ULONG nHandler = (ULONG)except_handler;
        PCHAR pTest = (PCHAR)0;
        __asm   {
                mov eax, FS:[0]
                push nHandler       //SEH 설치
                push FS:[0]
                mov FS:[0], ESP
        }
        __asm   {
                mov eax, 0          //SEH 발생
                mov [eax], 'a'
        }
```

```
        __asm   {
                mov eax, [ESP]      //SEH 제거
                mov FS:[0], EAX
                add esp, 8
        }
        return 0;
}
```

위의 코드는 SEH를 설치, 발생, 제거하는 순서로 구성되어 있다.

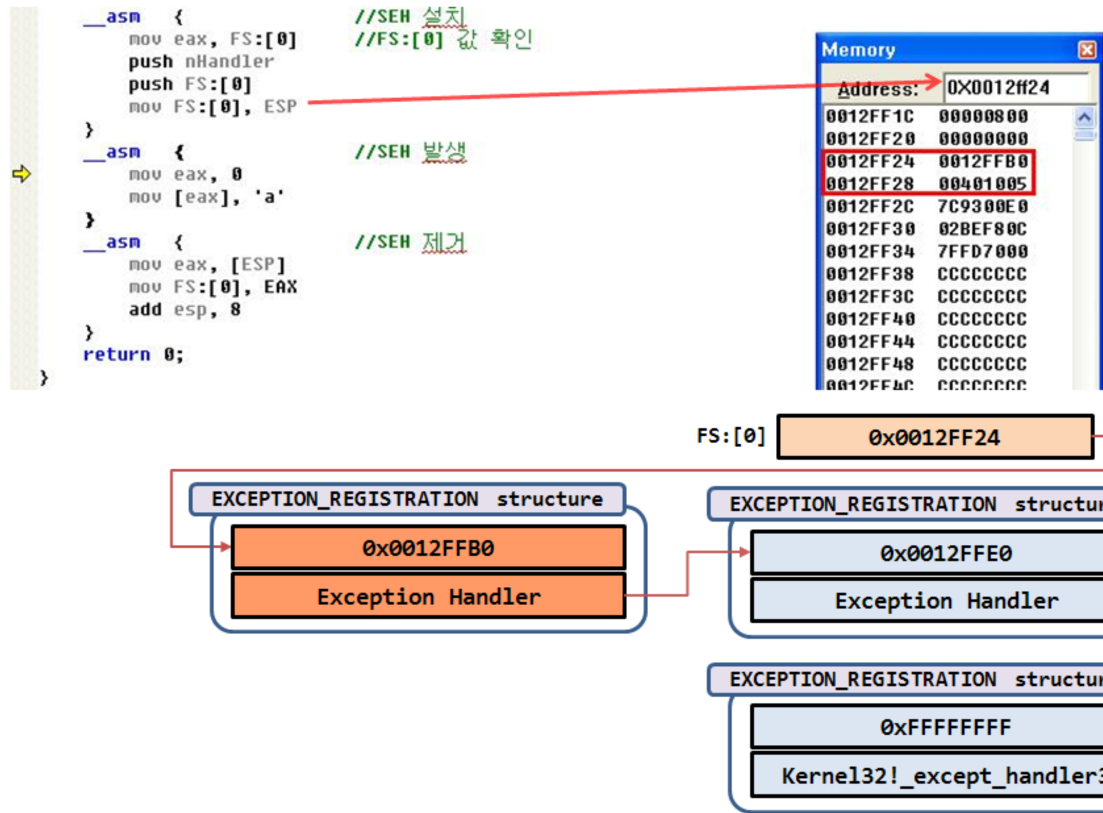
코드를 자세히 살펴보면 우선 SEH 구조체를 먼저 선언을 한다. 그리고 Exception Handler를 Push 한 후 FS:[0]를 Push를 해줌으로써 SEH 구조체를 하나 더 만들게 된다. 그리고 ESP의 값을 FS:[0]에 넣어 FS:[0]가 이 구조체를 가리키도록 한다.

SEH를 설치하기 전 레지스터 값과 메모리를 살펴보면 아래의 그림과 같다.

```
    void *ptr = EstablisherFrame;
    ContextRecord->Eax = (ULONG)&G_nValid;
    return ExceptionContinueExecution;
}

int main(int argc, char *argv[]) {
    ULONG nHandler = (ULONG)except_handler;
    PCHAR pTest = (PCHAR)0;
    __asm   {                   //SEH 설치
        mov eax, FS:[0]         //FS:[0] 값 확인
        push nHandler
        push FS:[0]
        mov FS:[0], ESP
    }
    __asm   {                   //SEH 발생
        mov eax, 0
        mov [eax], 'a'
    }
    __asm   {                   //SEH 제거
        mov eax, [ESP]
        mov FS:[0], EAX
        add esp, 8
    }
    return 0;
}
```

SEH를 설치한 후 레지스터 값과 메모리를 살펴보면 아래의 그림과 같다.



EAX에 0을 넣고 그 주소에 a를 넣어 예외를 발생시킨다. 즉, 잘못된 메모리의 주소에 a의 값을 넣으려고 하므로 예외가 발생하게 된다.

SEH를 제거할 때에는 설치과정과 반대로 ESP의 값을 FS:[0]에 넣어준다.
마지막으로 SEH를 제거한 후 레지스터 값과 메모리를 살펴보면 아래의 그림과 같다.

```
ULONG nHandler = (ULONG)except_handler;
PCHAR pTest = (PCHAR)0;
__asm   {                   //SEH 설치
    mov eax, FS:[0]         //FS:[0] 값 확인
    push nHandler
    push FS:[0]
    mov FS:[0], ESP
}
__asm   {                   //SEH 발생
    mov eax, 0
    mov [eax], 'a'
}
__asm   {                   //SEH 제거
    mov eax, [ESP]
    mov FS:[0], EAX
    add esp, 8
}
return 0;
}
```

5. 참고문헌

- Windows 구조와 원리 (OS를 관통하는 프로그래밍의 원리)