프로젝트별 상세 내용

프로젝트명	Context switching
개발환경	Windows 7
Lang/Tools	C/Source Insight 3.5, Visual studio 2013, 어셈블리/ml
사용기술	어셈블리를 이용한 CPU 레지스터 상태 저장/불러오기 코드 PE 포맷을 분석하여 실행 중인 프로그램에 실행 파일을 적재
프로젝트 소개	이 프로젝트에서는 이미 실행 중인 프로그램 안에 다른 실행파일을 적재시켜 실행시키는 프로그램을 작성하였다. 어셈블리 코드를 이용하여 CPU의 레지스터 상태를 전역변수로 저장하고 나중에 불러오면 레지스터 상태를 저장하던 그 부분으로 돌아갈 수 있는 함수, STST, LDST를 만든다. 그리고 이를 이용하여 프로그램 내부에서 다른 프로그램을 실행시키고 다시 원래상태로 돌아온다. 이 프로젝트를 완성시키기 위해 윈도우 실행파일의 구조와 어셈블리 언어를 공부하였다. 또한 메뉴 실행을 위해 메모리 맵 기법을 활용하며 함수 포인터에 대해 깊게 이해할 수 있었다.

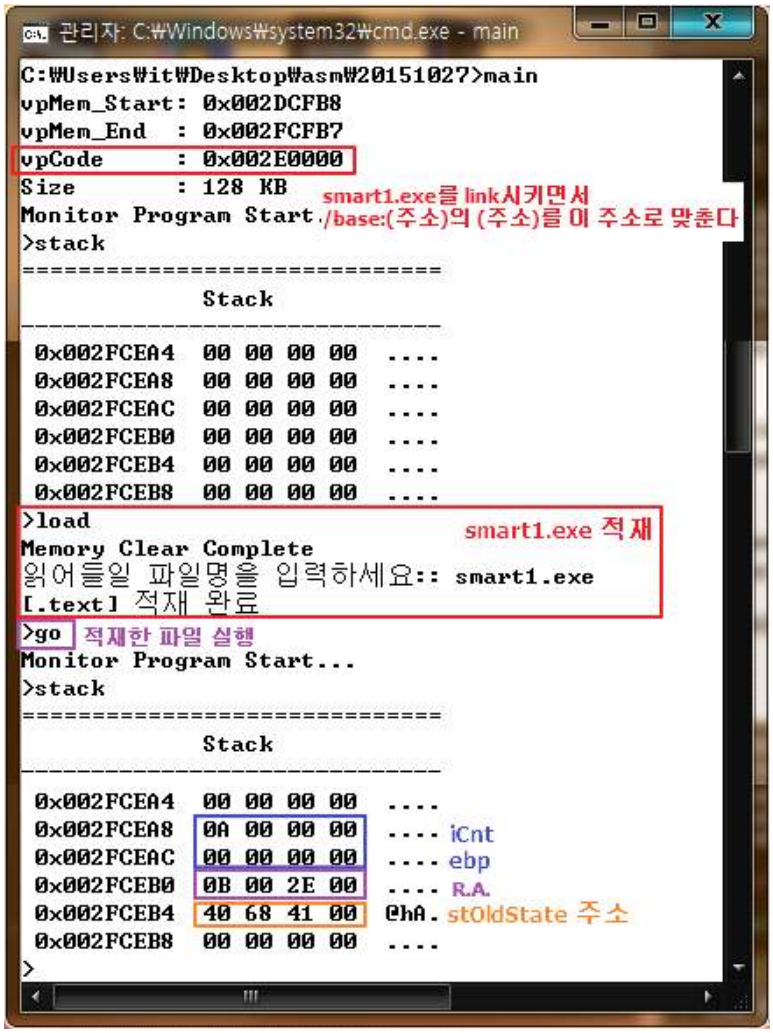

그림 1 프로그램 실행결과

어셈블리 함수 STST와 LDST

- STST: 어셈블리 함수/ STST를 호출한 시점에서의 CPU 레지스터값을 Context형 변수 stOldState에 저장
- LDST: 어셈블리 함수/ 인자로 들어온 Context 구조체의 값을 CPU 레지스터에 적용

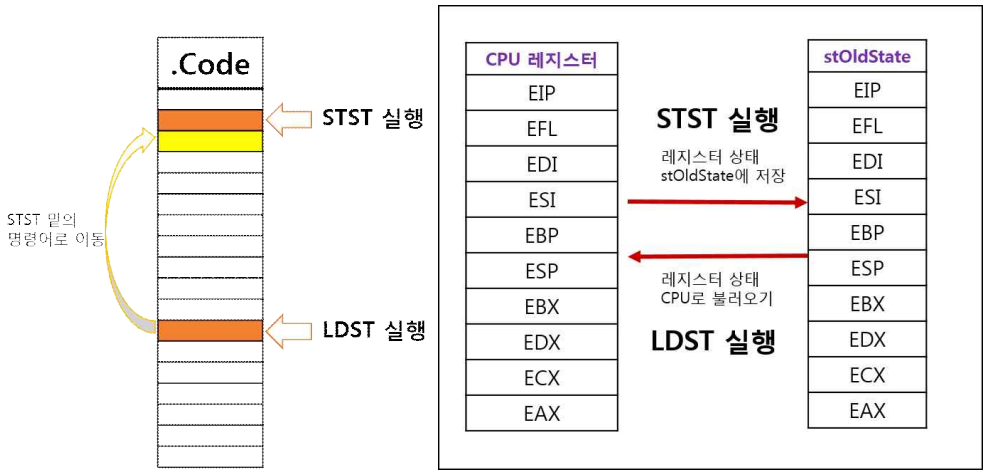

그림 2 STST-LDST 실행 시, 레지스터 상태 변화 및 명령어 실행점 변화

어셈블리 코드

STST

레지스터 상태를
Context형 변수에 저장

```
_STST        PROC NEAR32
        push ebp
        mov ebp, esp

        pushfd
        mov esp, [ebp+8]
        add esp, 40
        pushad
        push [ebp-4]
        push [ebp+4]
        add esp, 20
        mov eax, ebp
        add eax, 8
        mov [esp], eax
        push [ebp]

        mov esp, ebp
        pop ebp
        ret
_STST ENDP
```

LDST

Context형 변수를
레지스터에 불러온다

```
_LDST        PROC NEAR32
        mov esp, [esp+4]

        mov eax, [esp+20]
        pop ebx
        mov [eax-4], ebx

        popfd
        popad

        mov esp, [esp-20]
        sub esp, 4
        ret
_LDST ENDP
```

Context

레지스터를 저장할
Context 구조체 정의

```
typedef struct {
        unsigned int eip;
        unsigned int efl;
        unsigned int edi;
        unsigned int esi;
        unsigned int ebp;
        unsigned int esp;
        unsigned int ebx;
        unsigned int edx;
        unsigned int ecx;
        unsigned int eax;
} Context;
```

적재시킬 프로그램 만들기 – by 어셈블리

적재 프로그램(smart1.exe) 소스	프로그램 내부에서 실행시킬 프로그램

.CODE
_init:
push eax ; &oldState를 stack에 저장
call _smart
call _LDST ; 메인 프로그램의 처음으로 back
PUBLIC _init
END
	void smart(void)
	{
	int iCnt;
	for(iCnt=0; iCnt<10; iCnt++);
	return;
	}

적재 프로그램 링크 시 entry 주소를 메인 프로그램의 vpCode와 같게 한다

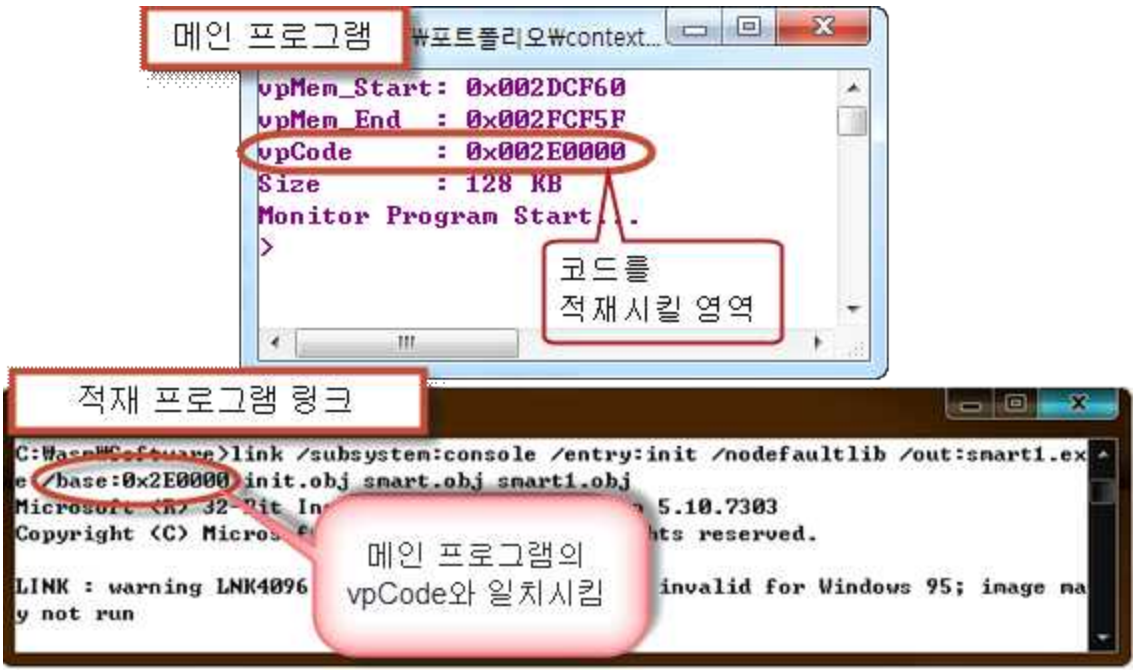

그림 3 적재 프로그램 컴파일 옵션 설정

윈도우 실행파일의 구조 분석 – PE 포맷

- 윈도우 실행파일의 구조를 분석하여 실행파일에 적재할 코드/변수 등을 읽어올 위치를 알 수 있다.

windows 실행파일 구조

영역 이름	크기		멤버	
DOS header	0x40 Byte		e_magic	매직넘버(도스는 MZ)
			e_lfanew:	NT header 위치 오프셋(실행파일 시작기준)
DOS stub	가변적			
NT header	가변적	4 Byte	Signiture	
		20 Byte	FileHeader	
		가변적	OptionalHeader	FileHeader,SizeOfOptionalHeader에 크기가 저장
Section header(.text)	33 Byte		Name	섹션 이름
			PointerToRawData	해당 섹션 데이터의 위치
Section header(.data)	"		"	"
Section header(.rsrc)	"		"	"
Section(.text)				code 영역 데이터, 프로그램 명령어 저장
Section(.data)				data 영역 데이터, 전역변수, 상수 저장
Section(.rsrc)				리소스 저장

헤더 구조체의 크기와 구조체 멤버들을 이용해 section header의 위치를 구할 수 있음

PointerToRawData에서 각 section의 시작주소를 알 수 있음

그림 4 윈도우 실행 파일 구조

프로그램 적재

- Code 영역의 시작 주소의 하위 2byte는 반드시 0이 되어야한다

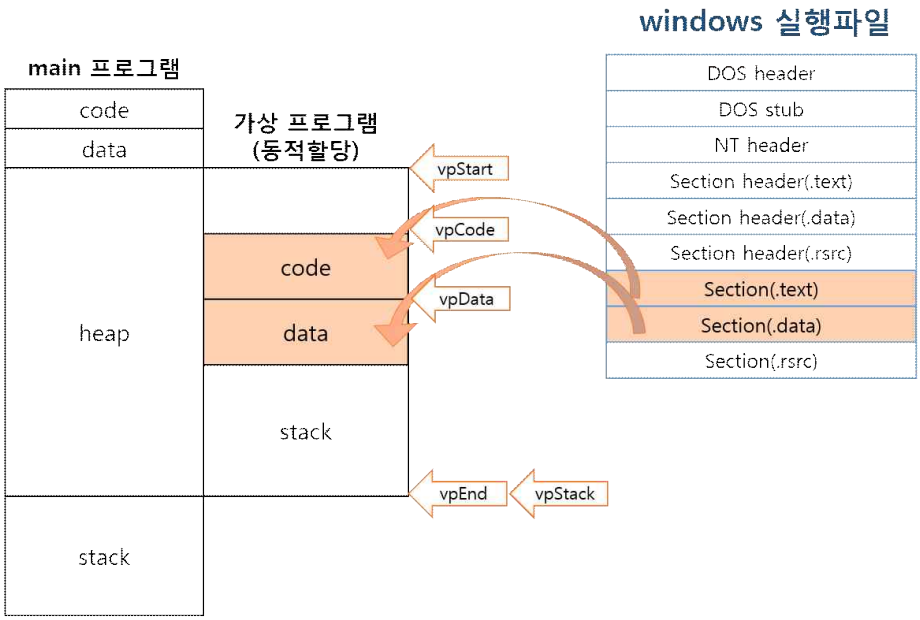

그림 5 실행파일을 main 프로그램의 동적할당 받은 영역에 저장

적재된 실행파일 실행 후, 다시 메인 프로그램으로 복귀하는 과정

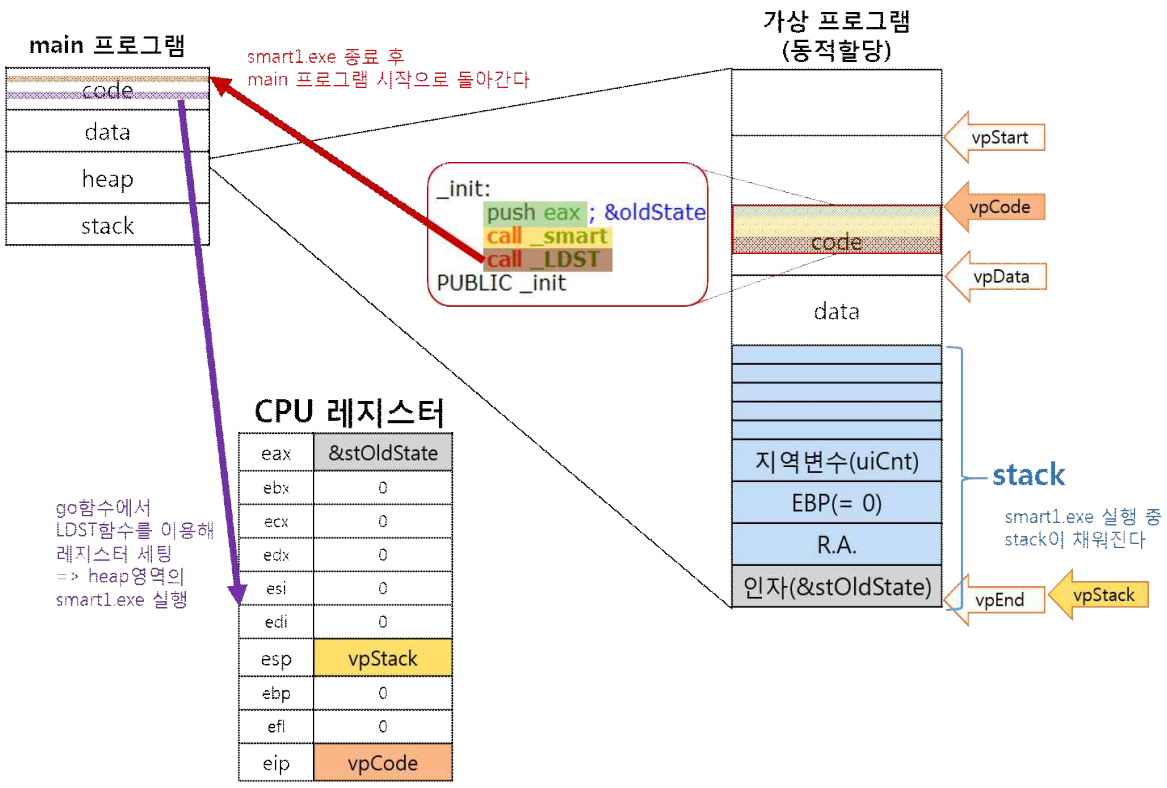

그림 6 적재시킨 smart.exe를 실행