이민규

1. 8086 Memory Architecture

시스템이 초기화 될 때 커널을 메모리에 적재한다. 현재 운영체제는 멀티태스킹이 가능하므로 메모리를 여러 세그먼트로 나누어 관리한다. 각각의 세그먼트에는 하나의 프로세스가 사용한다. 세그먼트는 stack-data-code 영역으로 나누어져 있다.

Code segment는 프로그램의 명령어들이 들어간다. 이 때 각 명령어들의 위치를 프로그램이 실행되기 전에는 정확히 알 수 없으므로 그 안에서는 논리적 주소를 사용한다. 그러므로 실제 명령어의 주소는 offset + logical address이다.

Data segment는 프로그램이 사용할 데이터가 들어간다. 여기서 말하는 데이터는 전역변수이고 이 세그먼트는 다시 네 부분으로 나뉘는데 각각 data structure, 데이터 모듈, 동적 생성 데이터, 공유 데이터 부분이다.

Stack segment는 지역변수가 들어간다. 멀티플 스택을 생성할 수 있고 각각 switch 가 가능하다. 스택에는 base pointer, stack pointer가 필요하다. bp는 현재 사용하는 스택의 가장 아래(고 주소), sp는 스택의 가장 위(저 주소)를 가리킨다.

2. 8086 CPU 레지스터 구조

레지스터는 여러가지 종류가 있다. 범용 레지스터, 세그먼트 레지스터, 플래그 레지스터, 인스트럭션 포인터로 구성되어 있다.

1. 범용 레지스터

범용 레지스터는 연산에 사용되며 eax, ebx, ecx, edx, esi, edi, ebp, esp가 있다. 세그먼트 레지스터는 code, data, stack segment를 가리키는 주소가 들어있다. 플래그 레지스터는 프로그램의 상태나 조건을 검사하는데 쓰인다. 인스트럭션 포인터는 다음 명령이 있는 메모리의 주소를 가리킨다.

16비트 cpu가 사용하는 범용 레지스터는 ax, bx, cx, dx, bp, si, di, sp이며 32비트 cpu는 앞에 e가, 64비트 cpu 는 앞에 r이 붙는다. ea~dx는 프로그래머의 필요에 따라 마음데로 사용해도 되지만, 다음과 같은 목적에 따라 사용하는게 일반적이다.

eax – 피연산자와 연산 결과 저장

ebx – ds 안의 데이터를 가리키는 포인터

ecx – 문자열 처리나 루프의 카운터

edx – IO 포인터

esi – ds가 가리키는 data segment 내의 어느 데이터를 가리키고 있는 포인터. 문자열 처리에서 source를 가리킴

edi – es가 가리키는 data segment 내의 어느 데이터를 가리키고 있는 포인터. 문자열 처리에서 destination을 가리킴

esp – ss가 가리키는 stack segment의 멘 꼭대기를 가리키는 포인터

ebp – ss가 가리키는 stack segment의 한 데이터를 가리키는 포인터

2. 세그먼트 레지스터

cs – code segment

ds, es, fs, gs – data segment

ss – stack segment

3. 프로그램 구동

```
void function(int a, int b, int c){
        char buffer1[15];
        char buffer2[10];
}

void main(){
        function(1,2,3);
}
```

위와 같은 프로그램을 실행할 때 레지스터가 어떻게 작동하는지 살펴본다.

위의 코드를 기계어로 변환해 그 코드가 code segment에 적재된다. 함수가 시작할 때 ebp와 esp를 새로 지정하는데 이것을 함수 프롤로그라고 한다. 이는 메인 함수가 시작할 때도, function이 시작할 때도 발생한다. function()을 시작 할 때를 보면 다음과 같은 코드가 있다.

```
<main+19>
push $0x3
push $0x2
push $0x1
call 0x80482f4

<function>
push %ebp
mov %esp, %ebp
sub $0x28, %esp
leave
ret
```

main+19에서 3, 2, 1순으로 스택에 값을 push한 후, function을 call한다. 이는 function의 인수를 반대로 스택에 집어 넣은 후, function을 호출하는 과정이다. function이 시작되면, 현재 ebp를 스택에 넣고, ebp를 esp까지 내린 후 char buffer를 위해 0x28만큼 esp를 내린다. buffer를 저장하기 위해서는 25바이트가 필요하지만 옛날 버전의 gcc에서는 word단위인 4바이트로 끊어 buffer1은 16바이트, buffer2는 12바이트만큼 필요하다. 하지만 2.96 이후의 버전에는 8바이트를 초과하는 버퍼는 4word단위로 할당해 두 버퍼에게 16바이트씩 할당된 것이다. 추가적으로 이 버퍼 사이에는 8바이트의 더미 값이 들어가 총 40바이트의 공간이 할당 된 것이다.

그 후 leave, ret을 수행하면 스택에는 변화가 없지만 ebp와 esp의 값은 이전 main함수에서 사용한 값으로 돌아간다.