**시스템보안**

이정현 교수님

**과제2**

소프트웨어학부 20163231 신승은

**1. 64비트(IA-32e) Register 구조 분석**

Register는 처리 중인 데이터나 처리 결과를 임시 보관하는 CPU내의 기억장치를 말한다.

프로그램 실행에 관련된 기본 register는 다음 4가지로 나눌 수 있다.

**(1) General Register**

크기별로 register를 분류하면 가장 작은 8비트 register는 AL(오른쪽 8비트 하위), AH(왼쪽 8비트 상위)로 나누고, 두 레지스터를 AX라는 16비트 register가 있다. 이런 방식으로 16비트 register는 BX, CX, DX가 있다. 32비트 register에는 앞에 Extended에서 따온 ‘E’가 붙어 EAX, EBX, ECX, EDX가 있다. 64비트 register는 앞에 ‘R’이 붙는다.

앞서 말한 register들은 범용 register로 논리 연산, 산술 연산에 사용되는 register다.

**(2) Address Register**

Address Register들은 ESP(스택 포인터), EBP(베이스 포인터)와 같은 top과 bottom을 가리키는 register와 ESI(소스 인덱스), EDI(위치 인덱스)가 있다. EIP(인스트럭션 포인터)는 다음에 수행될 인스트럭션의 주소를 저장한다. PC(program counter)라고도 한다.

**(3) Segment Selector Register**

Segment selector register는 모두 32비트가 아닌 16비트의 register로, 시작 주소를 저장하는 역할을 한다. CS(code segment),DS(data segment),SS(stack segment),ES,FS,GS가 있다.

**(4) Flag Register**

Flag register인 EFLAGS는 연산 결과 및 시스템 상태와 관련된 여러 가지 플래그 값을 저장하여 논리 연산을 확인할 때 주로 사용한다. 상태 플래그(s), 제어 플래그( c), 시스템 플래그(x)로 구성된다. 상태 플래그 중에서 CF(carry flag)는 부호 없는 산술 연산 결과가 목적지에 저장할 수 없을 만큼 큰 경우 1로 설정된다. PF(parity flag)는 연산한 결과가 1인 비트의 수가 짝수일땐 1로 유지되고, 홀수일땐 0으로 설정된다. ZF(zero flag)는 연산한 결과가 0일땐 1, 아닐땐 0으로 설정되는데, visual studio에서는 ZR로 표시한다. OF(overflow flag)는 부호있는 산술 연산의 결과가 너무 크거나 작아서 목적지에 저장할 수 없을 때 1로 설정된다. SF(sign flag)는 연산결과가 음수일 경우 1로 설정된다.

Kernel 시스템에 관련된 기본 register는 다음과 같다.

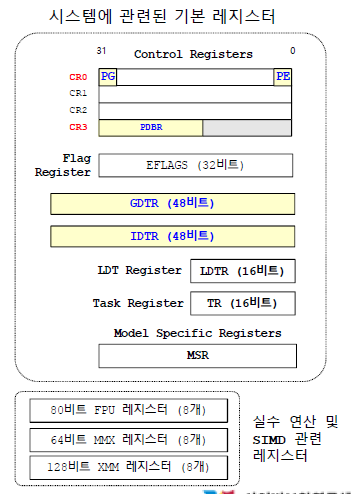


그림 kernel에 관련된 기본 register

우선 Control Register에서 CR0 register는 운영 모드를 제어하고, 리얼 모드에서 보호 모드로 전환하는 역할과 캐시, 페이징 기능을 활성화 시킨다. 대표적으로 PG(paging), PE(Protection Enable) 플래그가 있다. PG는 paging을 사용여부를 나타내고, PE는 보호모드 사용여부를 나타낸다.

CR3은 PDBR(페이지 디렉터리의 물리주소)를 16비트 설정하고, 나머지는 페이지 캐시에 관련된 기능을 설정한다.

GDTR은 global descriptor table register로 global descriptor table의 시작주소를 가지고 있다.

IDTR은 interrupt descriptor table register로 interrupt descriptor table의 시작주소를 가지고 있다.

**2. 64비트(IA-32e) 리눅스 부팅 과정 분석**

부팅은 PC가 켜진 후에 리눅스가 실행되기 전까지 수행되는 일련의 작업 과정을 의미한다.

1) 컴퓨터가 켜지면 CPU의 시작 주소인 0xffff0에서 BIOS가 실행된다.

2) BIOS가 ROM에 있던 IVT(interrupt vector table), BIOS 데이터, ISR(interrupt service routine)을 메모리에 로드한다. 이때 BIOS는 하드웨어를 초기화한다.

3) BIOS가 INT 0x19를 호출하는 ISR의 주소를 가르키도록 한다. 첫번째 섹터를 0x7c00에 로딩 한다.

4) 로딩 직후 부트 섹터가 자신의 코드를 0x90000번지로 복사한 뒤, 0x90000번지의 부트 섹터가 실행된다.

5) INT 0x13을 통해 셋업 코드를 0x90200번지에 로딩한다. 그리고 셋업 코드가 INT 0x13을 통해 커널 이미지(/boot/vmlinux-2.6.38-8-generic )를 0x1000000번지(=LOAD\_PHYSICAL\_ADDR)에 로딩한다.

6) 셋업 코드가 INT 0x13을 통해 초기 RAM 디스크 이미지를 임의의 상위 번지 메모리에 로딩하고 로딩된 주소와 크기를 vmlinuz의 부트 파라미터 영역에 기록한다.

7) 6)까지는 real mode에서 실행했다. protection mode로 전환한다. 전환하는 과정은 다음과 같다.

(1) A20 게이트를 활성화하기 위해 A20 게이트와 AND 연산을 통해 A20 비트 값을 결정한다. 이를 BIOS 인터럽트를 사용하여 결정한다. (INT 0x15)

(2) EFLAGS 레지스터의 IF 플래그(interrupt flag)를 0으로 설정하여 BIOS 인터럽트를 비활성화를 한다. 이젠 BIOS가 아닌 OS에서 만든 인터럽트 서비스인 IDT를 사용한다.

(3) CR0 레지스터의 PE(protection Enabled)를 1로 설정하여 활성화 시킨다.

(4) GDT와 GDTR 레지스터를 설정한다

- BOOT\_CS, BOOT\_DS, BOOT\_TSS 세그먼트만 설정하고 나머지는 커널 초기화 때 설정한다. Setup\_gdt()함수에 임시로 설정

(5) IDT와 IDTR 레지스터를 설정한다.

-IVT는 항상 0x00000번지에 고정되어 있지만, IDT는 초기 위치가 정해져 있지 않다.

-IDT 테이블 정보만 생성하고, 실제 엔트리의 정보는 커널 초기화 때 설정한다.

Setup\_idt()함수에 임시로 설정

8) 커널 이미지를 빈 메모리에 임시로 복사한 뒤, decompress\_kernel()함수(arch/x86/boot/compressd/misc.c)를 통해 압축 해제한 후 원래 위치로 옮기고 제어권을 커널로 넘긴다.

-decompress\_kernel()함수는 압축 해제를 하는 함수로 호출하는 곳에서 LOAD\_PHYSICAL\_ADDR로 정의가 되어있다.

9) 커널을 초기화하는 과정은 다음과 같다

- start\_kernel()이 호출되어 main.c에서 IDT 정보들을 설정한다. 그 다음 main.c에서Rest\_init()함수를 호출한다.

- rest\_init()에서 idle프로세스와 init프로세스를 설정한다. Kernel\_thread함수를 호출해 커널을 초기화 시켜준다. kthreadd프로세스의 PID=2다.

- idle 프로세스(PID=0) : 항상 대기중인 상태로 있다가 시스템의 원활한 동작을 위해 메모리에 올라가 있는 다른 프로세스들을 상태에 따라 swap in/out 시키는 역할을 담당한다. Init 프로세스의 부모 프로세스다.

-init프로세스(PID=1) : 모든 사용자 프로세스의 조상 프로세스다. 사용자 로그인 프로세스를 구동하고, 로그인하면 쉘을 생성한다.

10) 페이지 테이블과 디렉토리를 설정한 후에 페이지 디렉토리 베이스 주소를 CR3 레지스터의 PDBR에 저장한다.

11) CR0레지스터의 PG(paging)=1로 설정하여 페이징 모드로 전환한다. 커널 코드가 가상주소 0xc0000000에 있는 것처럼 작동하게 된다.

12) GDT와 GDTR 설정

- Global Descriptor는 64비트다. TYPE속성은 세그먼트의 타입 또는 데이터 세그먼트로 설정이 가능하다. S속성은 디스크립터 타입(1: 세그먼트 디스크립터, 0: 시스템 디스크립터)을 설정이 가능하다. DPL속성은 세그먼트의 권한(11: 사용자, 00: 커널)을 설정한다.

- 세그먼트 디스크립터의 타입(S=1)은 실행, 읽기, 쓰기, 역방향확장, 접근여부로 나누어 설정할 수 있다.

-시스템 디스크립터의 타입 중 대표적인 3가지 종류는 0x9의 32비트 TSS, 0xc 32비트 콜 게이트, 0xE의 32비트 인터럽트 게이트가 있다.

-인터럽트 게이트는 커널 루틴의 오프셋이 앞과 뒤 각각 16비트, 핸들러의 코드 세그먼트 셀렉터가 16비트가 특징이다.

-콜 게이트는 인터럽트 게이트와 같은 특징을 갖고 있고, 추가적으로 파라미터 개수를 표시하는 4비트를 갖고있다.

- GDTR에서 GDT의 base 주소를 가지고 CS의 인덱스 부분과 GDT 엔트리 크기를 곱한 값을 더하면 GDT 엔트리 주소가 나와 실행하여 세그먼트 셀렉터를 동작하게 된다.

(GDT base 주소 + index \* 8바이트(GDT 엔트리 크기) = CS의 GDT 엔트리 주소)

13) IDT생성

- GDT와 마찬가지로 RAM상에 저장한다.

- 256개의 디스크립터로 구성된다.

-0x20부터 사용자 정의 인터럽트 사용이 가능하다. 인터럽트 호출 과정은 다음과 같다.

- INT 0x80을 실행하면 사용자 권한으로 IDT 주소로 이동한다.

- 인터럽트 디스크립터에서 핸들러의 코드 세그먼트 셀렉터를 통해 GDT로 이동한다.

- 커널 권한으로 GDT의 base 주소로 이동한다.

- 인터럽트 디스크립터의 첫 번째 오프셋으로 핸들러 함수가 실행된다.

14) initrd는 루트 파일 시스템을 마운트할 준비와 실제 마운트하는 역할을 담당한다. 처리 과정은

-gzip으로 압축된 initrd 이미지를 ram디스크에 전개한다.

-dev/ram을 루트 파일 시스템으로 마운트한다. Initrd\_load()에서 진행한다.

-/linuxrc를 실행하여 필요한 커널 모듈의 로드와 진짜 루트 파일 시스템을 마운트하고 변환한다.