3.1 운영체제 개요

운영체제

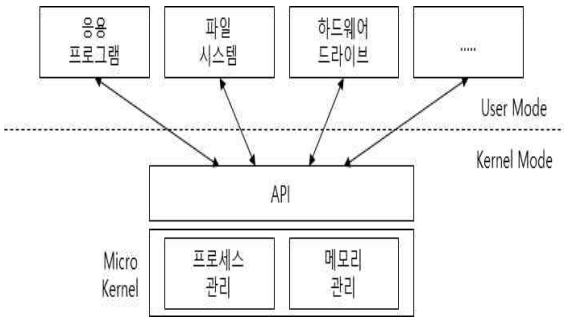
Memory, Process, Device, File 관리 후 H/W에서 이용

운영체제 구조

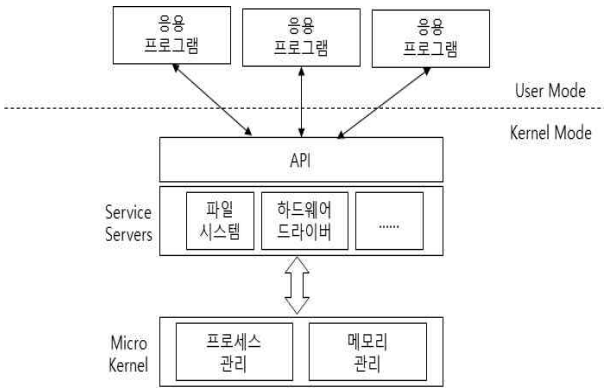
1. Monolithic Kernel
 * 다른 구성 요소와 통신 가능(호출)
   -> 높은 성능
 * 시스템에 제한 없이 접근 가능
 * 새 기능 추가를 위한 수정 및 유지 보수 어려움
 * UNIX / LINUX

2. Micro Kernel
 * 모듈 간의 통신이 많아 성능 감소

3. 윈도우, MacOS : Hybrid 형태



<Micro Kernel 구조>



< Hybrid 구조 >

3.2 80x86 프로세서 구조

프로그램(Instruction) 실행
 * Fetch - Decode - Execute

Processor 계열
1. CISC (Complex Instruction Set Computer)
 * 많은 수의 instruction들로 구성됨
 * Instruction : 복잡한 동작의 고수준 연산을 수행
 * Intel 80x86 , AMD 계열

2. RISC (Reduced Instruction Set Computer)
 * 적은 수의 짧고 간단한 intruction들로 구성됨

Byte Ordering

1. Big Endian

 * 0x12345678 ->

0x12	0x34	0x56	0x78

 * RISC 계열

2. Little Endian

 * 0x12345678 ->

0x78	0x56	0x34	0x12

 * CISC 계열 : Intel, AMD

3. Bi-Endian
 * 둘중 하나를 SW에서 선택

4. 문자열은 char 배열이기 때문에 1바이트씩 저장
   -> endian에 상관없이 동일함

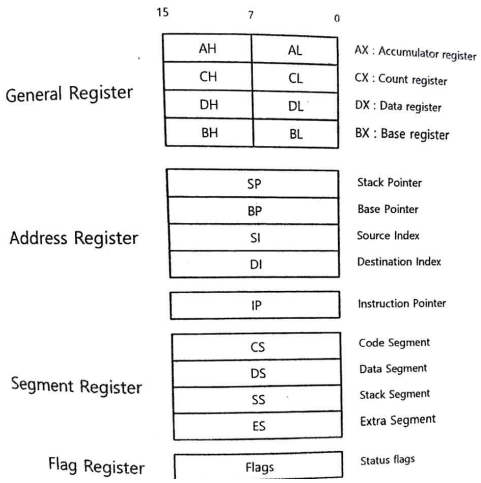
Process mode
1. real mode
 * multi process 불가능 = 단순 타이핑 기계
2. protected mode
 * 현재의 컴퓨터 프로세스 모델

x86 프로세서 종류
* 16비트 IA-16 (8086~80286)
* 32비트 IA-32 (80386~ 80686)
* 64비트 IA-32e
* IA-64는 Intel이 HP와 공동으로 개발한 IA-32와는 전혀 다른 새로운 Instruction set을 가지고 있는 아키텍쳐이다.
IA-32와 하위호환성이 없다.

3.3 Register

Register
 * 처리 중인 데이터나 처리 결과를 임시 보관하는 CPU 내의 기억 장치

8086 레지스터 (IA-16)



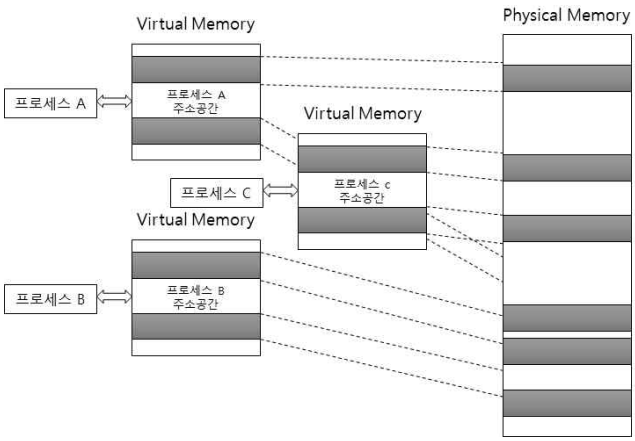
80386 레지스터(IA-32)

구분		이름	비트	용도
General Register	EAX	Accumulator	32	주로 산술 연산에 사용(함수의 리턴값 저장)
	ECX	Count Register	32	개수, 횟수 등을 저장하는 카운터로 주로 사용
	EDX	Data Register	32	Accumulator의 확장으로 EAX관련 연산에 주로 사용
	EBX	Base Register	32	일반 데이터 레지스터로 사용. 원래 16비트에서는 주소 레지스터로 사용했었음
Address Register	ESP	Stack Pointer	32	스택의 top 주소를 가리킴
	EBP	Base Pointer	32	스택 bottom 주소를 가르킴
	ESI	Source Index	32	문자열 연산에서 source 주소 저장
	EDI	Destination Index	32	문자열 연산에서 destination 주소 저장
	EIP	Instruction Pointer	32	다음에 수행될 instruction의 주소 저장 **PC(Program Counter)**라고도 함
Flag Register	EFLAGS	Flag Register	32	연산 결과 및 시스템 상태와 관련된 여러 가지 플래그 값 저장

구분		이름	비트	용도
Segment Selector Register	CS	Code Segment Register	16	code segment descriptor의 시작 주소 저장
	DS	Data Segment Register	16	data segment descriptor의 시작 주소 저장
	SS	Stack Segment Register	16	stack segment descriptor의 시작 주소 저장
	ES	Extra Segment Register	16	extra data segment descriptor의 시작 주소 저장
	FS	Extra Segment Register	16	extra data segment descriptor의 시작 주소 저장
	GS	Extra Segment Register	16	extra data segment descriptor의 시작 주소 저장 (예, stack guard의 random canary 저장)

3.4 Virtual Memory 관리

Virtual Memory = 사용자 프로그램의 메모리
* 사용자 프로그램을 여러 블록들로 분할
* 실행 시 필요한 블록들만 비연속적으로 주기억장치에 적재



커널 주소 공간 매핑

1. vmalloc 영역
 * Context Switching 불필요
 * 물리 메모리에서는 연속되지 않아도 되지만, 가상 메모리에서는 연속된 주소로 할당

2. pkmap 영역
 * Context Switching 필요

3. fixmap 영역
 * 고정의 가상주소 임의의 물리주소로 매핑하는데 사용 (용도가 이미 정해져 있음)

Virtual Memory & Physical Memory

1. Virtual Memory
 * 각 프로세스마다 할당하는 논리적 주소
 * 주기억장치의 용량보다 큰 메모리 용량을 주소 지정 가능
 * 프로세스가 실행될 때 필요한 일부분만 physical memory에 로드 함으로써 메모리를 절약할 수 있음

2. Virtual Memory를 Physical Memory로 주소 변환 과정에서 오버헤드가 발생

Memory Management Unit (MMU)

1. CPU가 Virtual Address를 생성함

2. MMU
 * VA를 PA로 변환해주는 하드웨어 모듈

3. Translation Lookaside Buffer (TLB)
 * 최근에 이용된 주소 변환 정보를 저장하는 캐시

Virtual Memory 관리 기법

1. Segmentation
 * 사용자 프로그램을 서로 다른 크기의 논리적인 단위로 분할

2. Paging
 * 사용자 프로그램을 같은 크기의 블록들로 분할

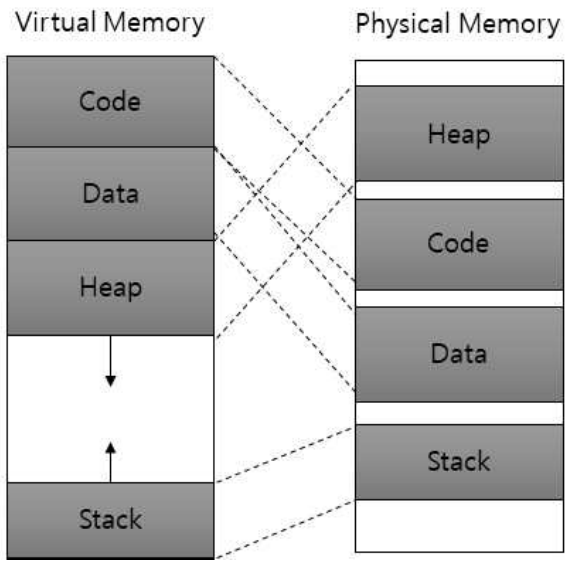
3. Segmentation & Paging 혼합 기법

Segmentation 기법

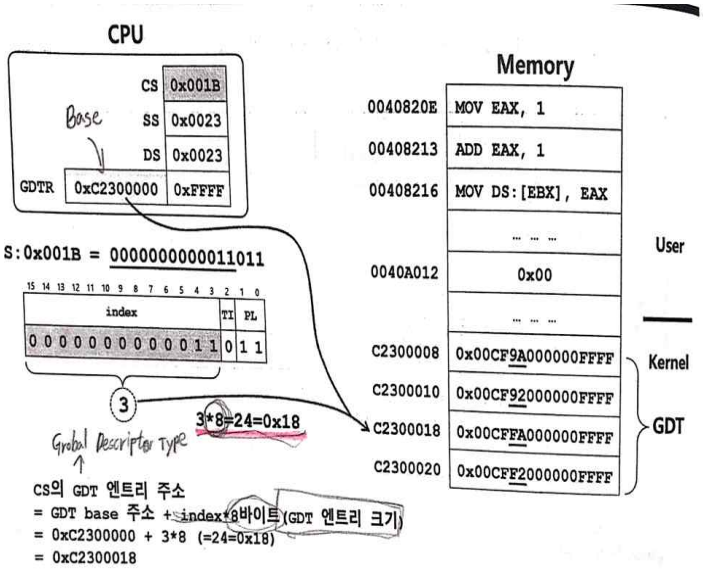
1. 프로그램을 블록 단위로 분할할 때 논리적인 개념 (코드, 데이터 등)을 가지고 서로 다른 크기의 블록들로 분할하는 기법

2. Segment : 분할되는 프로그램 블록들
 ex) Code, Data, Heap, Stack

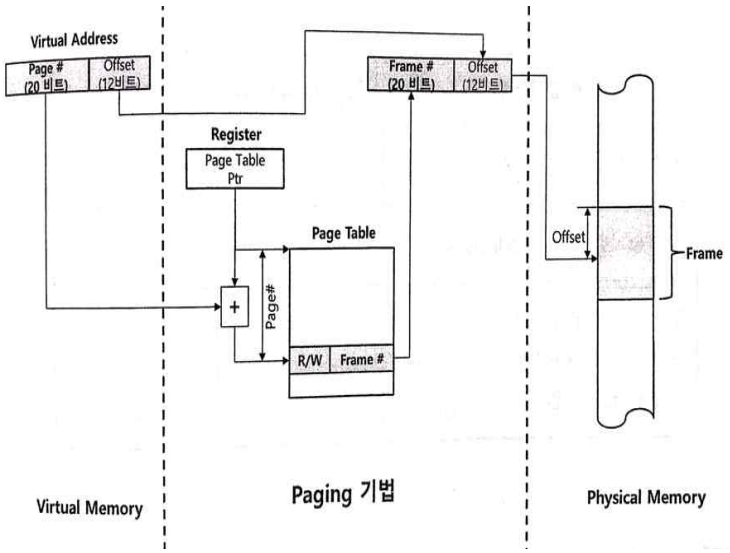
3. Segment 공유나 보호가 쉬움



Segment Selector 동작 과정



Paging 기법 주소변환



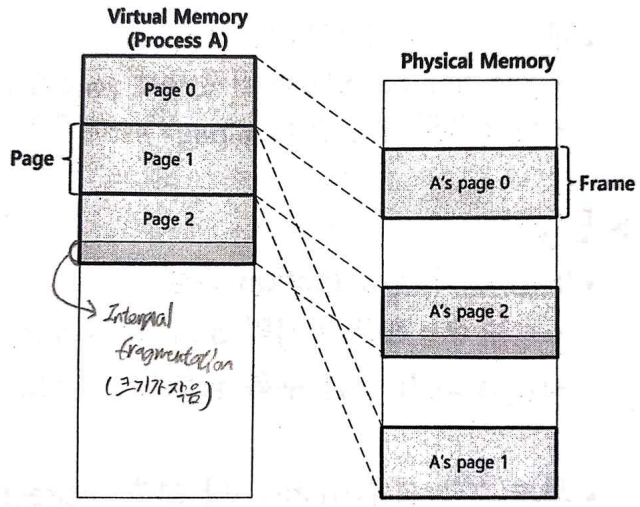
Segmentation 장단점

1. 장점
 * Segment별로 서로 다른 protection 기능 제공
   - read / write / execute 권한 부여
 * Segment 공유가 가능하고 재배치가 쉬움

2. 단점
 * Segment 크기가 클 경우 연속 메모리 할당이 어려움
 * external fragmentation 발생 : 메모리 낭비 초래
   -> limit 와 base+offset 사이의 공간

Paging 기법

1. Page : VM을 일정한 고정 크기로 분할하는 단위
2. Frame : PM을 일정한 고정 크기로 분할하는 단위
3. VM의 page들 가운데 현재 수행에 필요한 page만 PM에 적재
4. page를 적재할 때 연속적인 frame을 할당할 필요 없음
5. frame을 할당 받지 못한 page는 하드디스크에 저장
6. page table을 이용하여 page가 적재된 frame 위치 탐색



Paging 기법 장단점

1. 장점
 * 메모리 할당과 해제가 빠름
   - 할당 : free page list에서 첫 번째 page에 할당
   - 해제 : 해당 page를 free page list에 추가

2. 단점
 * Internal fragmentation 발생
   - 프로세스마다 마지막 page에서 fragmentation 발생
   - page 크기가 클수록 낭비되는 메모리 공간이 많아짐
 * 프로그램 공유나 보호가 어려움

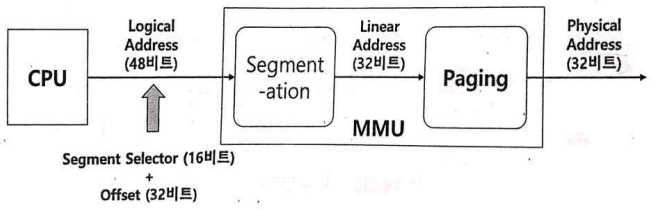
Paging과 Segmentation 비교

비교 항목	Segmentation	Paging
프로그램의 논리적 구조 정보 필요	yes	no
프로세스당 Virtual Address 공간	many (비연속할당)	1 (연속할당)
프로그램 간 코드 공유	yes	no
Fragmentation	External	Internal

Segmentation / Paging 혼합 기법

1. 두 가상기억장치 관리 기법의 장점을 모두 수용
2. 분할
 1) Segment 단위로 분할
 2) 각 Segment들을 page 단위로 분할
3. 로딩
 * 분할된 page 단위로 주기억장치에 로딩

80x86 주소 공간 매핑



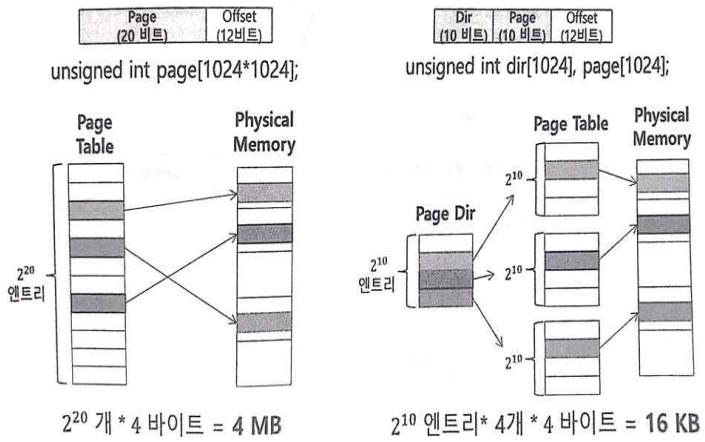
- 32비트 주소 버스를 통해 4GB Virtual Memory사용
- 16비트의 Selector와 32비트의 offset 주소(Virtual Memory)으로 이루어진 Logical Address를 CPU가 생성
- Segmentation은 4GB의 메모리를 segment 단위로 쪼갠 것으로, 32비트 Linear Address를 만듬

PDE와 PTE 구조

1. PDE (Page Directry Entry)
 * 12bit ~ 32bit : Page Table Base 주소
2. PTE (Page Table Entry)
 * 12bit ~ 32bit : Physical Page Base 주소

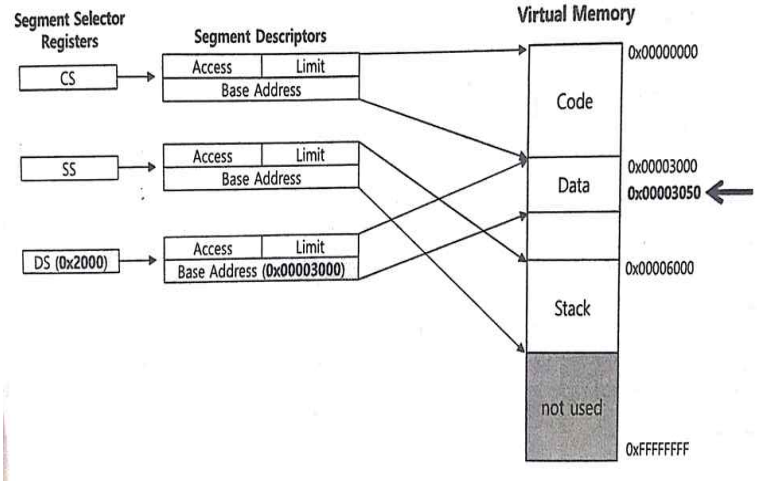
2단계 Paging 장점

1. 응용 프로세스가 3개의 세그먼트로 구성되어 있다고 가정
2. 한 프로세스당 주소 공간의 모든 page 개수 만큼의 page table 크기가 팔요

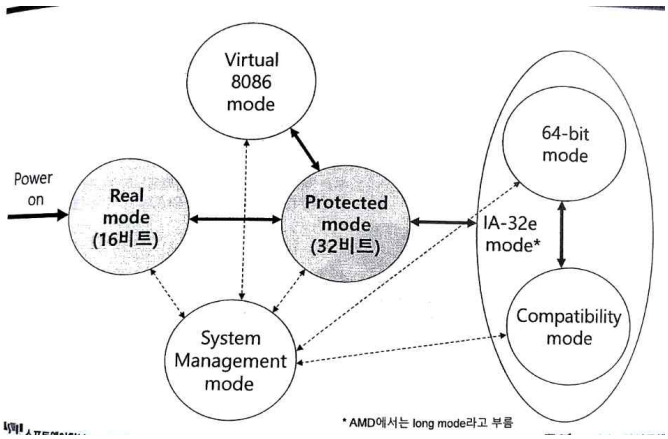


Multi-Segment Memory Model

1. Segment register에 설정된 값을 GDT의 인덱스로 사용됨
2. Segment Descriptors 내의 Base 주소가 Segment Base 주소로 사용됨



x86 동작 모드

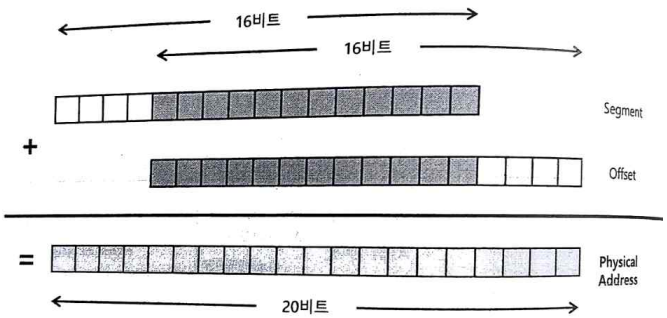


1. Real Mode

8086(IA-16) 호환 모드

20비트 주소 버스 사용 위해 16비트 Segment Register 사용

총 1MB의 메모리 사용 가능



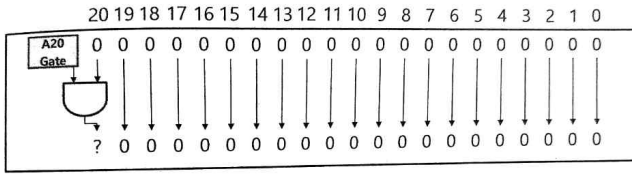
Segment Register와 Offset Register를 중첩시킨 20비트의 Physical Address를 생성

2. Protected Mode

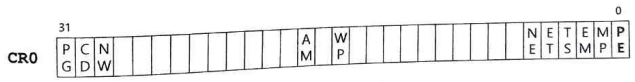
80286 이상에서 사용되는 동작모드, 메모리 보호 및 Segmentation, Paging 기능을 포함(286은 Paging 제외)

*** Protection 모드 전환 과정**

➢ A20 게이트 활성화



➢ CR0 레지스터의 PE(Protection Enabled)=1로 설정



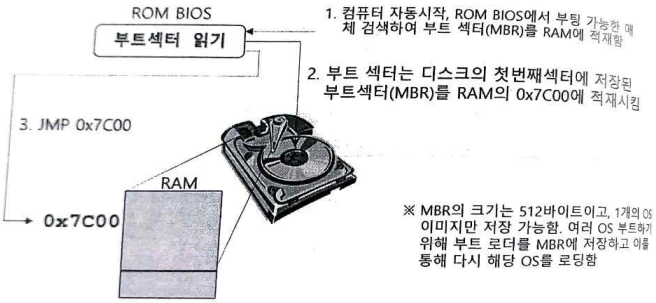
3. System Management Mode(SMM)

전원관리, 시스템 보안, 진단 등의 기능 수행

4. Virtual 8086 Mode

Protected 모드에서 Read Mode용 프로그램을 그대로 수행할 수 있도록 주소 변환을 하는 동작 모드

Booting Convention

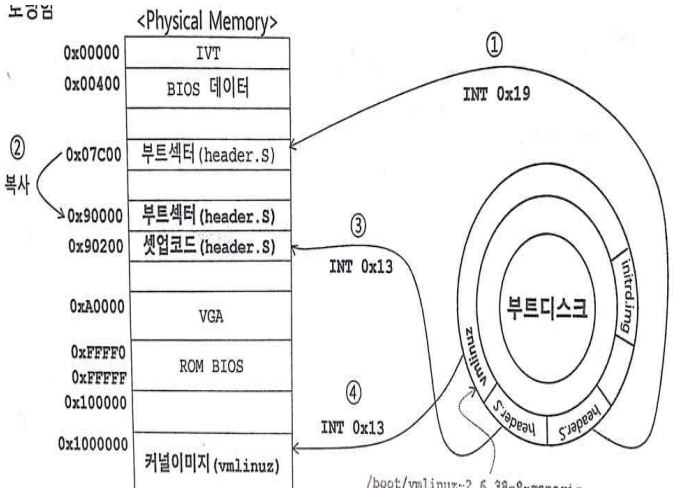


OS 측면 : 부트 섹터(MBR)는 디스크의 첫째 섹터 (1 또는 0)에 있다.

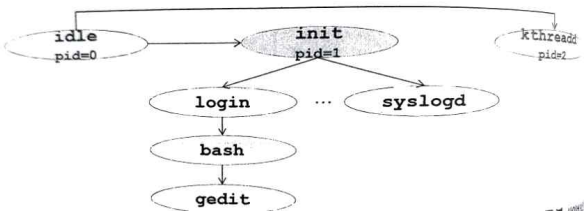
BIOS 측면 : 부트섹터는 메모리의 0x7c00번지에 로딩한다.

커널 이미지 로딩

1. BIOS가 INT (interupt) 0x19를 호출하여 첫 번째 섹터를 0x07c00번지에 로딩함 -> 로딩 직후 부트섹터가 자신의 코드를 0x90000번지에 로딩함
2. INT 0x13을 통해 셋업코드를 0x90200번지에 로딩
3. 셋업코드가 INT 0x13을 통해 커널 이미지를 0x1000000번지 로딩



idle 및 init 프로세스의 실행



idle 프로세스 (PID = 0)

- 항상 대기중인 상태로 있다가 시스템의 원활한 동작을 위해서 메모리에 올라가 있는 다른 프로세스들을 상태에 따라 swap in/out 시키는 역할을 담당
- Swapper 프로세스 라고도 부름

init 프로세스 (PID = 1)

- 모든 사용자 프로세스의 조상 프로세스
- 사용자 로그인 프로세스를 구동하고 로그인 하면 쉘을 생성함