[Chapter 1]

경영학과/2018100233/김정민

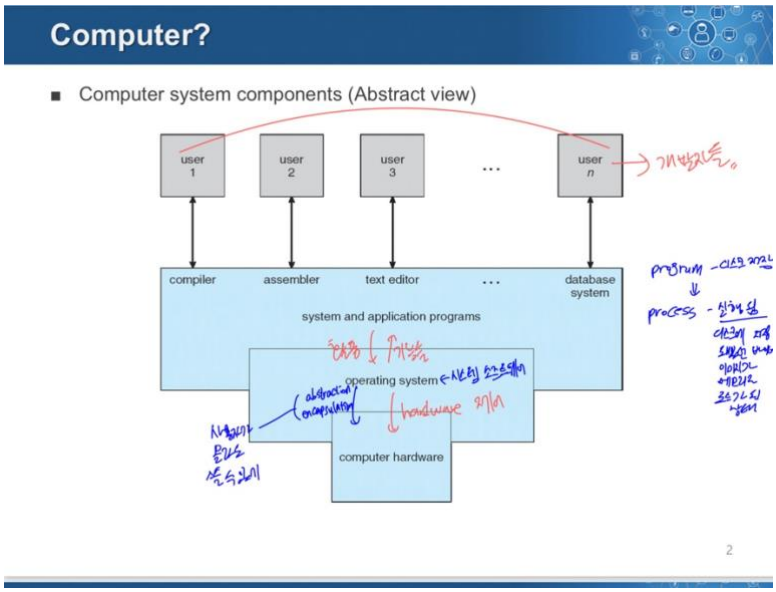
I. Computer

A. 컴퓨터의 종류

1. 개요: 컴퓨터는 사용 용도에 따라서 크게 세 가지로 분류된다.
2. Server Computer
   - 고성능 하드웨어 및 대용량 저장소를 가진 컴퓨터
   - 서버 역할 수행 (웹 서버 등)
   - OS : 유닉스 (60%), 윈도우 (20~30%), 리눅스 (10%)
3. Personal Computer (PC)
   - 랩탑 등 개인이 사용하기 위해 만들어진 컴퓨터
   - OS : 리눅스, 윈도우, 맥 (국산은 쓰지 않는 편)
4. Embedded Computer
   - 냉장고, 인공위성, 슈퍼컴퓨터, 자동차, 항공기 등에 들어가는 컴퓨터
   - OS : 대표적으로 VxWorks (hard real-time)
5. 참고
   - OS가 없으면, 기능적으로 컴퓨터라고 할 수 없다.
   - 연산을 위해서는 대상과 행위가 필요하다. 이를 행하는 주체는 CPU (프로세서)다.
   - 우리나라에도 서버 소프트웨어를 만드는 회사가 있다? -> 티맥스 소프트

B. 컴퓨터 구성요소 및 구조

1. Hardware -> Operating System -> System and Application programs -> User 순으로 상위 개념



2. CPU, disk controller, memory 등 구성요소들끼리 버스 형태의 네트워크를 구축해서 데이터를 주고 받는다.
   - CPU : 메모리에서 명령을 가져와서 각 controller에게 명령을 내린다.
   - Disk controller : CPU에서 보낸 명령이 디스크로 직접 전달되는 것이 아니라, 중간에서 컨트롤러에 의해 처리된 후 디스크에서 명령을 수행한다.
   - USB controller : CPU에서 보낸 명령을 처리한 후에 각 USB 디바이스로 신호를 보내서 명령을 수행한다.
   - Graphics adapter : CPU에서 보낸 명령을 받아서, 그래픽 처리와 같은 작업을 수행한다. (그래픽 카드) 명령을 처리해서 모니터를 통해 데이터를 출력한다.
3. Modern PC architecture
   - Memory Controller Hub(MCH)와 I/O Controller Hub(ICH)로 나뉜다.
     - 두 부분은 PCI BUS로 위 아래를 연결한다. (중추 역할)

- MCH
  - 메모리와 연결되어서 데이터를 처리하는 부분
  - CPU와 System Controller(Northbridge)는 연결되어 있고, Northbridge는 AGP (그래픽 카드)와 DRAM(메인 메모리)에 연결되어 있다. 각각 연결된 라인을 AGP Bus, Memory Bus라고 부른다.
  - 속도에 민감하다.
  - MCH는 AGP, DRAM과 데이터를 주고받으면서 처리하기 때문에 데이터의 양이 많고 처리속도가 빨라야만 한다.
  - 따라서 대역폭이 크고 빠르다.
- ICH
  - 속도에 비교적 민감하지 않다.
  - Peripheral Bus Controller(Southbridge)는 ISA Bus, USB, Dual EIDE에 연결되어 있다. 이외에는 LAN, SCSI, BIOS 등이 있다.
  - Disk Controller : 디스크를 연결하는 Bus의 대역폭 -> Disk 자체가 수행 속도가 느려서 버스의 대역폭이 빠를 필요가 없다. 빨라봤자 병목이 발생한다.
  - 어차피 버스 속도보다 I/O 디바이스의 속도가 더 느리다.

II. Operating Systems (운영체제)

A. OS란?

1. Resource manager (하드웨어 리소스를 관리한다.)
   - Abstraction (encapsulation)
     - 사용자는 하드웨어가 어떻게 돌아가는지 몰라도 쉽게 사용 가능
   - Sharing
     - Time multiplexing : cpu는 1개인데, 여러 개의 프로세스가 돌아갈 때, 동일한 단위 시간에는 한 개의 프로세스만 실행할 수 있다. -> 단위 시간마다 어떤 프로세스를 실행할 지에 대한 시간 분배를 해준다.
     - Space multiplexing : 메모리를 나눠서 사용하거나 공유한다.
   - Protection : 한 프로세스가 cpu를 독점하는 경우를 방지한다.
   - Fairness : 여러 프로그램이 공정하게 자원을 사용할 수 있도록 분배한다.
   - Performance : 상위 개념들을 지키는 와중에 최선의 성능을 발휘한다.
2. 다른 용어
   - Resource allocator
   - Control program
   - Kernel
     - OS의 기능들 중 핵심적인 부분들만 지칭
   - 광의의 OS : OS가 제공하는 (없어도 되는) 커널 위의 시스템 프로그램들까지 포함
3. SW which manages Computer HW resources for
   - Convenince
   - Efficiency

III. Computer System Operation

A. Computer HW architecture

1. CPU, I/O 디바이스, 메모리 등이 Bus 형태의 네트워크로 연결되어 있다.
2. CPU
   - ALU (Arithmetic Logical Unit)

        - PC (Program Counter) : 다음에 실행할 명령어의 주소를 저장하는 레지스터
        - IR (Instruction Register) : 현재 실행 중인 명령어를 저장하는 레지스터
    - CU (Control Unit)
        - SP (Stack Pointer) : 프로그램 실행 중에 사용되는 스택(메모리의 일부)의 현재 위치를 가리키는 레지스터
        - PSW (Process Status Word) : CPU를 사용하고 있는 프로세스의 상태를 비트 단위로 저장하는 레지스터
    - MMU (Memory Management Unit) : 연산 수행 X
        - General purpose registers : 함수 콜이 일어난 경우 파라미터의 값을 해당 레지스터에 저장해놓고 가져온다.
3. Memory
    - 상위부터 나열
        - Stack : 프로그램 실행 중에 함수 호출 및 반환, 임시 데이터 저장 등에 사용된다. LIFO의 특성을 가진다.
        - Heap : 동적 메모리 할당을 위한 영역이다.
        - 상수 영역 : 프로그램 내부에서 수정할 수 없는 상수 데이터를 저장하는 영역이다.
        - 코드 영역 : 프로그램의 실행 코드가 저장되는 영역이다. 프로그램(exe 파일 등)이 실행되면 해당 프로그램의 코드 덩어리가 코드 영역에 로드된다.

B. CPU operation

1. Von Neumann architecture
    - 위에서 정리한 방식(하나의 메모리 안에 코드와 데이터를 같이 저장해놓는 방식)이 폰 노이만 아키텍처다.
    - 하버드 아키텍처에 비해 최적화가 용이하다.
2. Harvard architecture
    - 코드와 데이터를 따로 저장하고 각각을 CPU에 연결한다.
    - 병렬적이기 때문에 더 빠른 성능을 보인다.
    - 하지만 코드에 따라서 메모리를 설계하기가 어렵다는 단점이 있다.
    - 한눈에 보면 폰 노이만보다 좋아 보이지만, 최적화 문제에 있어서 안 좋다.
3. Instructions
    - Arithmetic : add, subtract 등
    - Logical : and, or, not, shift 등
    - Control flow : goto, if, call, return 등
    - Data : load, store, move, input, output
        - Input, output은 I/O 디바이스에 적용되는 인스트럭션이다.
4. XIP architecture
    - 플래시 메모리에 코드 덩어리를 저장하고 직접 실행하는 구조다. (하버드 아키텍처와 다른 점)
        - ROM의 일종
    - 추가적인 RAM 부담 줄어들고, 실행 중에 프로그램 코드를 별도로 로드할 필요가 없다.
    - Cahce hit rate가 엄청나게 상승한다.

C. Bootstrapping in Linux (부팅 절차)

1. CPU가 제일 먼저 구동되면서 자신의 상태를 체크한다.
2. BIOS에 있는 코드 시작지점 (0xfffffff0)의 주소에 있는 인스트럭션을 실행한다.
    - 피지컬 메모리 체크

3. BIOS (Basic Input/Output System) / UEFI (Unified Extensible Firmware Interface) 실행
   - BIOS
     - 하드웨어 (메모리, I/O 장치 등)를 확인 및 초기화한다.
     - ROM에 구현된다.
   - UEFI
     - BIOS보다 업그레이드된 부트 매니저 제공한다.
     - 플래시에 구현된다.
     - 펌웨어 업데이트 및 GUI 환경을 제공한다. -> 호환성, 패치, 사용자의 BIOS 설정 더욱 용이
4. BIOS와 UEFI가 부팅 장치를 찾고, 부트로더를 불러온다. (LILO / GRUB)
   - LILO (Linux Loader)
     - 리눅스 시스템에 특화된 부트로더
     - 유연성 부족, 새로운 하드웨어와 파일 시스템에 대한 지원 한계 -> GRUB
   - GRUB (Grand Unified Boot Loader)
     - 다양한 운영체제와 파일시스템에 대한 지원을 제공하는 유연한 부트로더
     - 사용자에게 다양한 부팅 옵션 제공
     - 다중 부팅 환경 지원
   - BIOS에서는 MBR (Master Boot Record)을 사용
   - UEFI에서는 ESP (Efi System Partition)을 사용
5. BIOS와 UEFI가 부트로더를 로드하고 실행한다. (주도권을 넘긴다.)
6. 부트로더는 압축된 커널을 로드한다.
   - 압축된 커널을 스스로 압축을 해제하고, 해제된 커널로 주도권을 넘긴다.

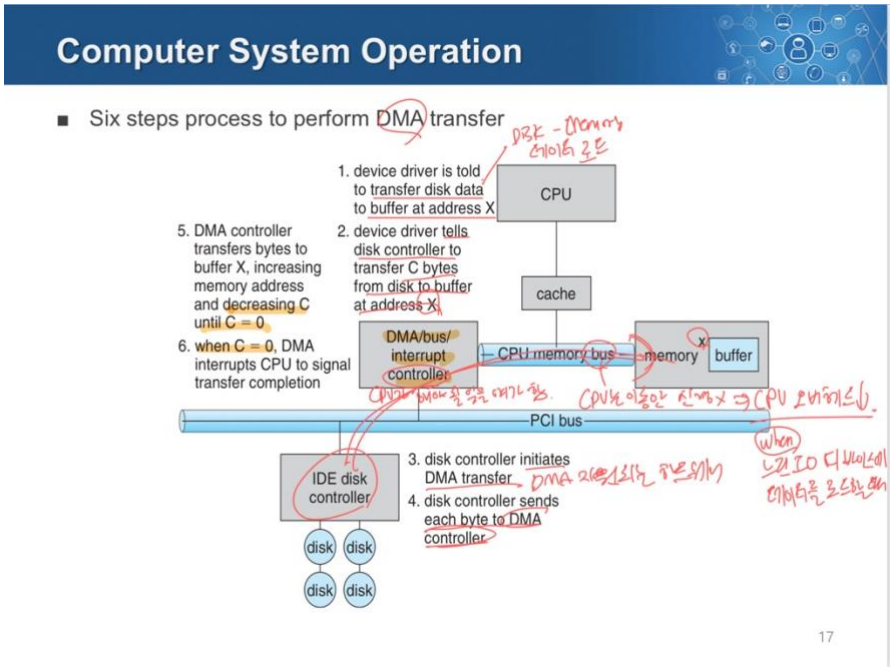
D. CPU 구조 (ISA에 따라 구분)

1. ISA (Instruction Set Architecture)
   - CISC (Complex Instruction Set Computer)
     - 복잡한 구조
     - 자주 사용되는 명령어 조합을 합쳐서 여러 동작을 수행한다.
     - 클럭은 더 들어도 속도는 빠르다.
   - RISC (Reduced Instruction Set Computer)
     - 단순한 구조로 성능도 챙긴다.
     - 하나의 명령어로 하나의 동작만 수행한다.
     - 어떻게 성능을 챙기는지?
       - 하드웨어 성능 증가
       - **Pipelining**
2. Pipelining
   - 명령어를 Fetch, Decode, Execute, Write Back 단계로 나누어서 최대 4개의 명령어를 동시에 수행 가능
   - 하지만 4배만큼 성능 향상이 이루어지진 않는다.
     - 각 인스트럭션의 단계마다 실행시간이 다르기 때문에 idle time 발생
     - 각 인스트럭션 간에 의존성이 존재하면 파이프라이닝이 완벽하게 이루어지지 않음
     - 조건문이 있으면, 다음에 수행할 명령어가 무엇인지 모름 (적중률 낮아도 손해는 없다.)
3. ILP (Instruction-Level Parallelism)
   - Superscalar (intel)
     - CPU에서 하드웨어적으로 ILP 수행
   - VLIW (DSP)

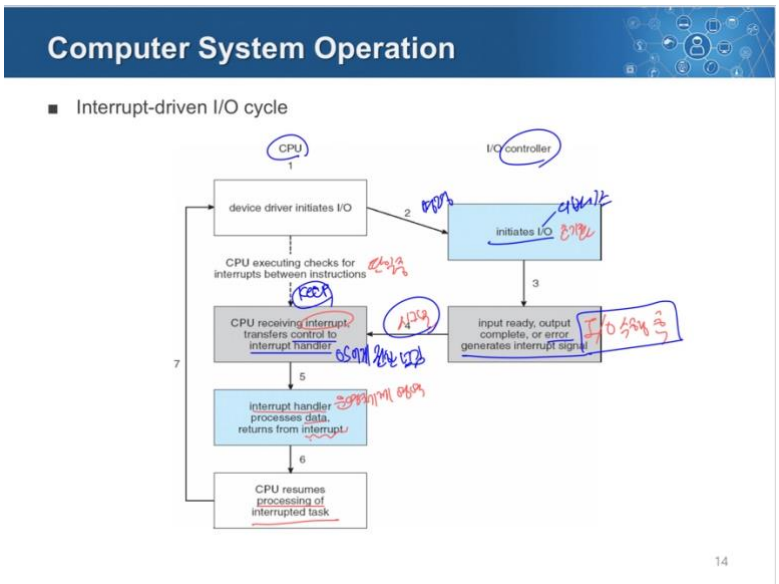
    - 소프트웨어적으로 ILP 수행 (컴파일러 레벨)
- Simultaneous multithreading
    - 단일 코어에서 여러 개의 쓰레드를 병렬 수행
- Multi-core
    - 여러 개의 물리적 코어가 하나의 칩에 포함되어 병렬 수행
    - 하나의 메모리를 여러 개 코어가 공유
- SMP (Symmetric multiprocessing architecture)
    - 여러 개의 CPU가 각각 레지스터와 캐시를 가지면서 하나의 메모리를 공유한다.
    - Parallel computing 성능은 떨어진다.
- NUMA (Non-Uniform Memory Access)
    - 한 시스템 안에서 메모리를 쪼개서 여러 개의 CPU가 각각의 물리적 메모리를 가지고, 서로 연결된 구조다.
    - Ex. 기상청 슈퍼 컴퓨터
- Clustered system architecture
    - Parallel : 한 시스템 안에서 동시 진행
    - Distributed system : 떨어져있는 컴퓨터들이 일을 쪼개서 분산 -> 네트워크로 연결 (interconnect)
    - 컴퓨터마다 메모리가 따로 있고, 공유 스토리지가 또 따로 있다.

E. I/O operation

1. CPU 명령(I/O instruction)을 받아서 I/O 디바이스를 동작한다.
    - Direct I/O
        - 컨트롤러를 통해 디바이스한테 CPU가 직접 명령한다.
    - Memory-mapped I/O
        - 메모리 특정 번지에 매핑 시켜서 메모리에 명령을 쓴다.
    - I/O 컨트롤러에 있는 레지스터들과 통신한다.
        - IR(Instruction Register), DR(Data Register)
2. I/O method
    - Programmed I/O
        - CPU가 I/O 디바이스와 직접 통신하면서 프로그램을 통해 입출력을 제어한다.
    - Interrupt
        - 하드웨어로 구현
        - 입출력 작업이 끝나거나 이벤트가 발생했을 때, I/O 컨트롤러가 CPU한테 신호를 보내는 방식
        - CPU는 다른 작업을 수행하다가 컨트롤러로부터 인터럽트를 받으면, I/O 동작이 끝났다는 것을 인지한다.
        - 운영체제 코드 중 특정 인터럽트에 대한 이벤트를 핸들링하는 코드 수행한다.
        - 폴링 : CPU가 디바이스에 계속해서 작업이 끝났는지 물어보는 것인데, 효율이 너무 좋지 않고 CPU 자원을 낭비한다.
    - DMA (Direct Memory Access)
        - CPU를 거치지 않고 입출력 데이터를 메모리로 전송하거나 메모리에서 입출력 디바이스로 보낼 수 있다.
        - 성능 향상

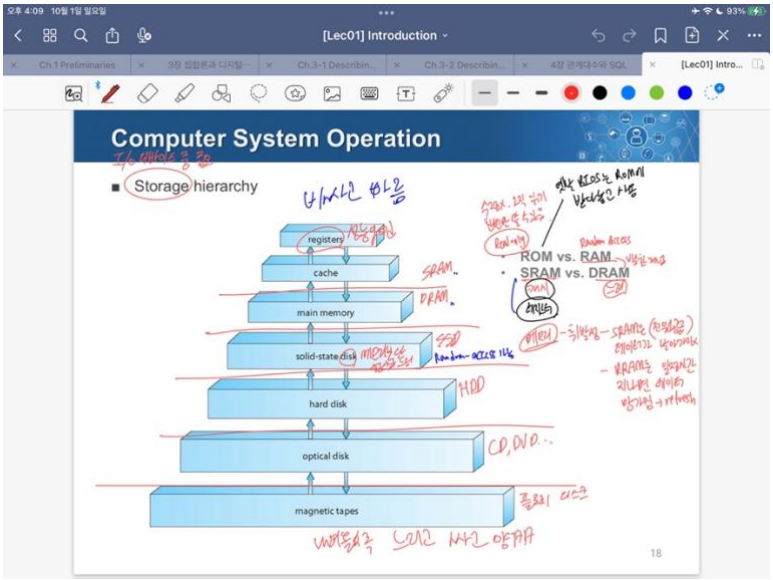


3. 인터럽트 종류.
   - Interrupt
     - 하드웨어 인터럽트 (비동기적)
     - Ex. Timer interrupt, keyboard interrupt
   - Trap
     - 소프트웨어 인터럽트
     - 프로세스가 I/O에 요청한다.
     - Ex. System calls (기능 호출, 프로세스 중단 -> 운영체제 코드가 CPU로)
   - Fault (Exception)
     - CPU가 스스로 거는 인터럽트
     - Ex. Divide-by-zero, page fault, protection fault



F. Storage (I/O 디바이스 중 중요)

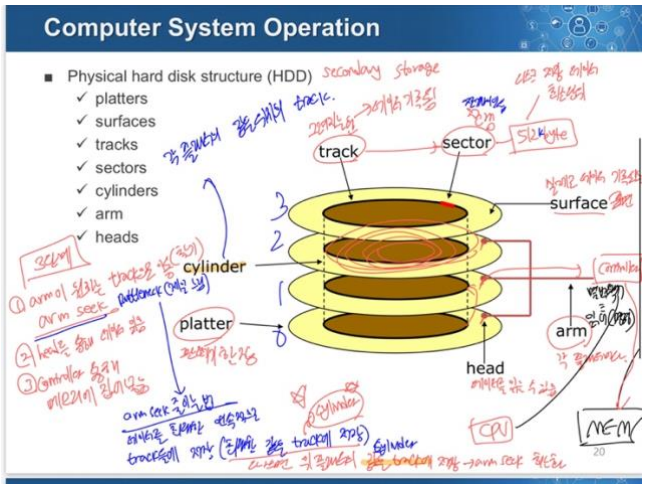
1. 개요 (한눈에 보기)



- Memory hierarchy
  - 아래로 내려올수록 느리다.
  - 하지만 disk부터는 비휘발성 -> 그래서 필요하다.

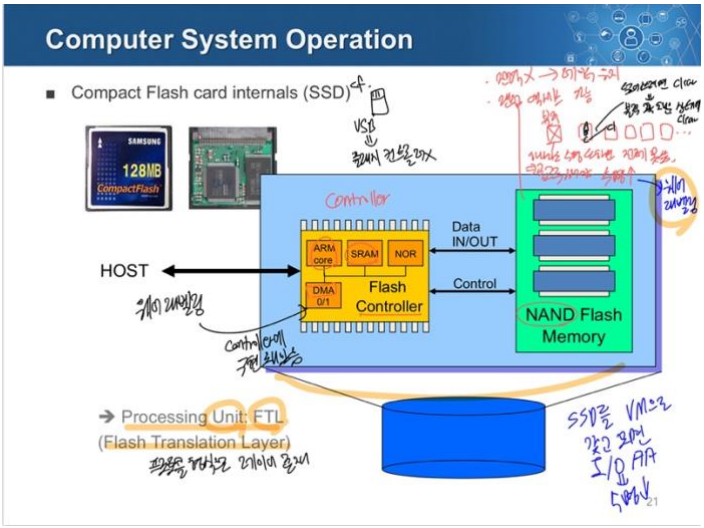
2. Caching
   - 로컬 캐시 데이터가 바뀌면, 메인 메모리, 디스크 등의 데이터도 변경되어야 한다. (**Cache Coherency**)
   - 이 경우 두 가지 방법이 있다.
   - Write-through
     - 값이 바뀔 때마다 바로 대체
     - 메모리와 디스크 간에 I/O 하느라 병목이 발생한다. -> 성능 하락
     - 그래도 일관성 면에서는 좋다.
   - Write-back
     - 일단 디스크는 변경하지 않았다가 CPU랑 버스가 유휴 상태일 때 (혹은 캐시가 해제될 때), 바뀐 데이터들을 한 번에 반영한다.
   - 하버드 아키텍처는 코드와 데이터 메모리가 분리되어 있기 때문에, 캐시 데이터가 대체되는 주기가 더 느려서 cahce hit rate가 올라간다.

3. HDD (Secondary storage)
   - 원하는 메모리 찾는 3단계
     - Arm이 원하는 track으로 이동 (arm seek) -> 여기서 병목 발생 (제일 느린 단계)
     - Head를 통해 데이터를 읽는다.
     - 컨트롤러를 통해 메모리에 집어넣는다.



- Arm seek 줄이는 법
  - 데이터를 최대한 연속적으로 track들에 저장
  - 최대한 같은 track, cylinder에 저장

4. SSD



- NAND Flash Memory에서 각 블럭은 하나라도 수명을 다 하면 전체를 사용할 수 없다.
  - 골고루 사용해야 메모리 수명을 늘릴 수 있다.
  - 블럭을 사용할 때 최대한 블럭을 꽉 채운 상태에서 Clear 시킨다.
  - 웨어라벨링이란 기술을 통해 가능하다.
  - 전력 공급 없이도 데이터가 유지된다.
  - 랜덤 액세스도 가능하다.

5. System call
- User mode와 kernel mode로 나뉘어져 있다.
- User mode
  - 프로그램이 수행되는 상태
  - 프로세스에서 OS한테 원하는 동작을 부탁한다. (trap)
- Kernel mode (OS)
  - OS가 CPU 사용권한을 받아서 OS가 동작하는 모드
  - 트랩(SW 인터럽트)을 트랩 핸들러한테 받아서 해당 동작을 수행해서 프로세스에 전달한다.

6. Hardware Protection
- 한 프로그램이 독점하는 경우를 방지한다.
- CPU protection
  - 한 프로그램이 CPU 계속 점유하는 것 방지
  - Timer interrupt (주기적으로)
- Memory protection
  - MMU를 통해 가능
  - Protection fault
- I/O protection
  - Dual mode (user, kernel)
  - 커널모드만이 수행할 수 있는 명령어들이 있다.
  - User mode (mode bit = 1), kernel mode (mode bit = 0)

7. OS가 동작하는 세 가지 상황
- Bootstrapping

- System calls
- Interrupts
- 아무것도 안해도 동작하는 경우가 있긴 하다. (Timer interrupt)

IV. Computer History

A. 1세대 (1945-55)

1. 폰 노이만의 Eniac
   - 강의실 6개만한 크기에 진공관과 플러그 보드를 연결해서 만든 컴퓨터
   - OS, 프로그래밍언어, 어셈블리어 존재 X
   - 진정한 컴퓨터라고 보기는 힘들다.

B. 2세대 (1955-65)

1. 기존의 진공관에서 트랜지스터로 변화
   - HW 성능 증가 -> 복잡한 프로그램 수행
2. Batch systems
   - Resident Monitor (운영체제다운 운영체제는 아님)
   - I/O 병목 때문에 CPU를 제대로 활용하지 못한다.
   - 프로세스 스위치가 일어나지 않고, 하나의 프로세스를 쭉 수행하고 다음 프로세스를 수행한다.

C. 3세대 (1965-80)

1. Integrated Circuits (집적회로)
   - 하나의 칩 안에 트랜지스터 집적도 증가 -> 성능 향상 -> SW 가용성 증가
2. IBM System/360 family (개인용 컴퓨터) 등장
3. Multiprogramming systems
   - I/O를 하는 중에 CPU가 다른 일을 할 수 있다. (Switching)
   - CPU 활용률 증가
4. Time-sharing systems
   - I/O가 없는 경우엔 한 프로그램이 CPU를 독점?
   - "No." timer를 통해 독점을 방지한다.
   - 응답속도 증가

D. 4세대 (1980-)

1. LSIs & VLSIs
2. Microprocessors
   - 프로세서가 더 작아지고 빨라졌다.
3. Storages
   - 더 크고 빨라졌다.
4. CPU work is offloaded to I/O devices.
   - CPU가 할 일을 I/O 장치로 전가해서, CPU 부담을 줄인다.
5. 진정한 Personal computers 보급
6. 현대적인 OS 특징
   - 커널만 가지고 있는 게 아니라 다양한 시스템 소프트웨어를 제공
     - GUI (Graphical User Interface)
     - Multimedia
     - Internet & Web
     - Networked / Distributed

V. Computing Environments

A. 다양한 컴퓨팅 환경

1. 전통적인 컴퓨팅
   - Mainframe system
     - Batch system – 한 사람만 동시 사용 가능, 개인용 X
     - Multiprogramming system
     - Time-sharing system
   - Desktop system (PC)
2. 모바일 컴퓨팅
   - Hand-held system (한 손으로 쥐고 다닐 수 있을 만큼 작다.)
     - 제한된 메모리
     - 느린 프로세서
     - 작은 출력 스크린
3. Real-time embedded computing
   - Hard real-time
     - 데드라인 안에 반드시 일을 해야 하는 시스템
     - Ex. 항공기 OS
   - Soft real-time
     - 데드라인 못 지켜도 치명적인 결과를 초래하지는 않는다.
     - Ex. 티켓팅
4. Client-server computing
   - 데이터 센터의 큰 규모의 서버
   - 분산 서버
     - 병렬 + 분산 + 저장소
5. Peer-to-peer computing
   - 클라이언트들끼리 연결되어 통신하는 형태
   - Ad-hoc
6. Cloud computing
   - Iaas (infrastructure as a service)
     - 하드웨어 자원만 제공
     - OS, DB, application 등 설치해야 사용 가능
   - Paas (platform ~)
     - 웹 서버 구축 -> 플랫폼, OS, 미들웨어 수준까지 구축
   - Saas (software ~)
     - Storage
     - 사용자는 그냥 사용하기만 하면 된다.
     - Ex. Icloud
   - **4차산업혁명**
     - **IoT**
     - **빅데이터 : 방대한 데이터 가공**
     - **클라우드 : 빅데이터 연산 (컴퓨팅 자원)**
     - **인공지능 : 연산 자원이 남으니까 가능한 기술**
7. Virtualization
   - VMM (Virtual machine manger)을 통해 가능

- 하나의 하드웨어, OS 위에서 독립적으로 다른 OS를 올릴 수 있다.
- Windows -> VMWare, Mac OS X -> Parallels
- OS마다 시스템 콜이 다르기 때문에 VMM이 필요하다.

8. Distributed computing
   - 여러 시스템이 서로 분산된 채로 연결되어 있는 형태
   - Ex. Hadoop

VI. OS History

A. 예전

1. IBM OS/360 : Multiprogramming (최초)
2. MIT CTSS (Compatible Time-sharing system)
3. MIT, Bell Labs, GE, MULTICS
4. Unix (1969)

B. Unix (1997-)

1. 유닉스를 기반으로 한 여러 가지 OS 탄생
   - Sun Solaris, Linux, Mac OS X. 등
2. OS마다 시스템 콜이 다 다르다.
3. 호환성 문제 어떻게 해결?
4. POSIX라는 유닉스 표준을 만들었다. (시스템 콜이 모두 같아서 호환성 제공)

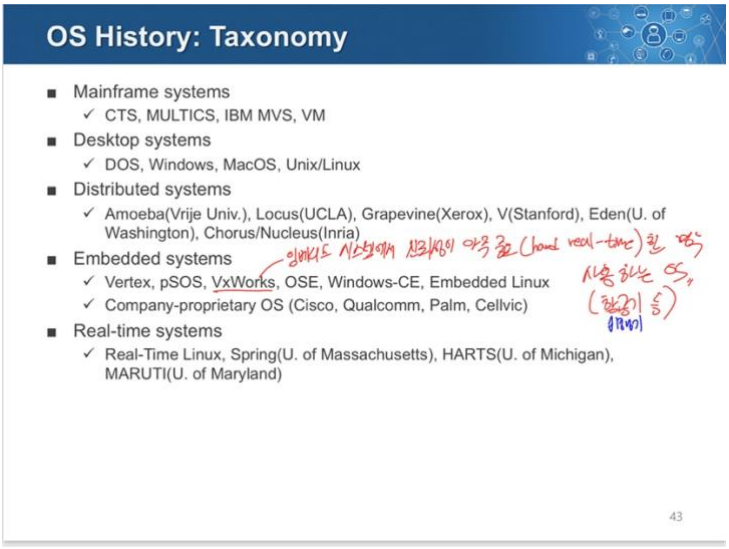
C. Windows

1. A 버전 : Kernel version
2. B 버전 : Major version
3. C 버전 : Minor version
   - 테스트 버전이라고 보면 된다.

D. Linux

1. 1983년 GNU 프로젝트에서 탄생
   - gcc, gdb, glibc 등 다양한 툴들 발생
2. 서버, 임베디드, 모바일 디바이스, 휴대폰 등에 사용되었다.
   - 안드로이드 플랫폼 또한 리눅스 커널 기반이다. (사실상 OS는 아니다.)
3. TV, 라우터, 포스기 등등에 이용된다.
4. 진정한 Personal computers 보급
5. 현대적인 OS 특징
   - 커널만 가지고 있는 게 아니라 다양한 시스템 소프트웨어를 제공
     - GUI (Graphical User Interface)
     - Multimedia
     - Internet & Web
     - Networked / Distributed

E. 시스템 분류에 따른 OS 사용



1. "하나만 꼭 알아야 한다면?" : VxWorks
   - 임베디드 시스템에서 꼭 알아야하는 OS
   - Hard real-time에서 사용하는 OS
   - 항공산업, 비행기 등에 사용