**[Chapter 1]**

소프트웨어융합학과/2020105742/한지훈

1. 컴퓨터란 무엇인가
   A. 컴퓨터는 연산을 하는 CPU와 피연산자 저장을 하는 메모리 등으로 이루어진 하드웨어를 구성요소로 가지면서 소프트웨어를 작동시킬 수 있는 모든 것
   B. 분류
      I. 서버렉: 원격의 다수 클라이언트로부터 요청을 빠르게 처리하기 위해 존재하며, 서버 컴퓨터라고 부르기도 함
      II. PC: 개인 사용자가 범용으로 쓰기 위한 것. 문서편집기, 게임기, 동영상 재생기 등을 범용적으로 사용 가능
      III. 임베디드 시스템: 태블릿, 스마트 워치, TV 등이 있으며, 이들의 공통점은 특정 목적을 위해 설계되었다는 것

2. 운영체제란
   A. 소프트웨어를 실행하기 위해서는 하드웨어 부품이 필요하다. 우리가 1+1과 같은 연산을 쉽게 할 수 있는 이유는 나 대신 하드웨어 자원 관리를 효율적으로 잘 해주는 근간이 있기 때문이다. 이 근간을 운영체제라고 할 수 있다.

3. 네트워크 인터페이스
   A. 네트워크 인터페이스는 버스구조로 되어있으며, CPU, 입출력 장치, 메모리 등 모든 요소가 공유한다. 네트워크 인터페이스의 특징은 1타임에는 1통신밖에 안된다는 것이다. 만약 CPU, disk가 동시에 메모리를 사용하고 싶어서 전기신호를 동시에 보내면 신호가 합쳐지고, 이 합쳐진 신호는 분류가 불가하다. 이 때문에 한 번에 한가지 통신밖에 하지 못하는 것이다.

4. modern PC architecture
   A. Northbridge
      I. 메모리, CPU, 그래픽카드 간에 아주 많은 communication
      II. 상대적으로 고성능을 발휘
      III. 메모리와 CPU는 당연하고 그래픽 또한 4k 화면 등 기술의 발전으로 인해 요구 data 량이 점점 커지는 추세
   B. Southbridge
      I. Northbridge에 비해 상대적으로 저성능
      II. LAN, SCSI: PCI 버스와 직렬로 연결 > 상대적으로 좋은 성능
      III. BIOS(usb, 키마, ...): PCI 버스와 병렬로 연결 > 상대적으로 낮은 성능

5. OS의 역할
   A. 주 목적: 컴퓨터의 하드웨어 자원을 나 대신 편리하고 효율적으로 관리하는 것
   B. 세부 역할
      I. Abstraction: 사용자가 복잡한 부분을 알지 않아도 됨
      II. Sharing: 멀티태스킹 시 하드웨어 자원을 공유하는 기법
      III. Protection: CPU나 I/O 디바이스 보호를 통한 안정적인 운영을 해 줌
   C. 전권: OS는 이 역할들을 수행하기 위해 제어, 할당 등 하드웨어 운영 관련 전권을 모두 가지고 있음

6. Computer HW architecture(abstract)
   A. CPU: 동작(프로세스)를 담당하는 부분
      I. ALU(arithmetic and logical unit): 연산 관련 명령어를 처리해주는 담당
      II. CU(control unit): 함수 사용 등 control flow가 바뀔 때 이를 조작하는 명령어를 처리

B. 레지스터/메모리

I. PC(program counter): 다음에 수행될 instruction의 위치를 저장

II. IR(instruction register): 수행되는 명령어 코드가 저장되는 곳

III. SP(stack pointer): 스택을 어디까지 쓰고 있는지 알려줌

IV. PSW(program status word): 프로세스 상태 값을 저장

7. Architecture 변천

A. 폰 노이만 Architecture

I. 구성: CPU와 메모리로 구성되어 있는데, 이때 data와 code segment가 메모리에 같이 올라와 있음

II. 단점: 순차적으로 data, code를 불러와야 하기 때문에 느리고 병목현상이 일어남

B. 하버드 Architecture

I. 구성: 메모리를 data만 담는 부분, code만 담는 부분으로 분리하여 속도를 높이고자 함

II. 단점: 메모리를 분리하는 설계상 cost가 크고, 메모리를 효율적으로 사용하지 못함

III. 사용처: embedded system(code 메모리를 flash로 만들어 사용)

IV. XIP(Execute in Place) 구조: 디스크를 거치는 것이 아니라 메모리에서 바로 실행시키는 것

8. 컴퓨터 부팅 순서

A. CPU 동작 시작

I. 세부 동작: CPU가 정상인지 초기화 검사작업 진행

II. 완료되면 상숫값 주소(0xFFFFFFF0)로 이동(바이오스 시작주소)

B. 바이오스 코드 실행

I. 세부 동작: 메모리, GPU 등이 동작하는지, 하드웨어 컨디션 등을 체크

II. 완료되면 MBR(Master Boot Record)를 통해 Boot loader의 시작주소를 찾는다.

C. Boot loader 실행

I. 세부 동작: 사용자의 선택/우선순위에 따른 OS 이미지를 실행시켜 줌

II. 디스크에 있는 커널 이미지의 압축을 풀고 메모리로 올려 실행시켜준 뒤, CPU 권한을 커널에 넘김

III. 종류: LILO/GRUB(기능 더 많음)

D. 커널의 CPU 권한 획득

I. 로그인 화면 볼 수 있음

II. 이후에도 알맹이 이외 부가적인 파일들을 불러오느라 잠깐 버벅거릴 수 있음(최근 컴퓨터는 거의 없음)

9. CISC, RISC, Pipelining

A. Instruction Set Architecture(ISA)

I. CISC(complex)

- 개요: 사람들이 복잡한 연산을 편리하게 하기 위해 customize > instruction 수, CPU 복잡도, 전력소모량 증가

II. RISC(reduced)

- 개요: CISC의 단점을 막기 위해 과거로 회귀했고, 그 결과 전력/시간 효율적으로 동작했지만 성능이 크게 떨어짐

B. Pipelining

I. 개념: CPU instruction 중 서로 dependency가 없는 것들은 최대한 중첩해서 실행하자

II. Dependency 없는 instruction: fetch, decode, execute, write back

III. 효율: 이론적으로 4배의 효율을 보여야 하지만 실제로는 약 2.3~2.5배 정도의 효율을 보임

IV. 효율 떨어지는 이유:
- 4중첩이 되는 case가 적어 pipeline이 깨지기 쉬움
- Instruction간에 ms 단위로 시간 차이가 날 수 있음
- 이전 Instruction의 결과가 나와야 다음 Instruction을 시작할 수 있는 경우

V. ILP(Instruction level parallelism)
- VLIW(SW를 통해 컴파일 타임에 동시에 수행해도 되는지 확인)
- Superscalar(HW를 통해 CPU 자체에서 병렬수행 가능 여부를 분석)

10. HW architecture 종류

A. Multi-core architecture: 하나의 CPU 안에서 여러 개의 core이 하나의 memory를 공유하는 구조

B. Symmetric multiprocessing architecture: 여러 개의 독립적인 CPU가 하나의 memory를 공유하는 구조

C. NUMA multiprocessing architecture: CPU뿐만 아니라 memory도 독립적으로 존재하는 구조

I. 특징: 성능이 매우 높음

II. 사용처: 기상청 슈퍼컴퓨터

D. Clustered system architecture: 네트워크로 컴퓨터 여러 개를 하나로 묶어 사용하는 구조

I. 특징: 컴퓨터 하나가 망가져도 서비스 퀄리티만 떨어지고 동작은 해서 안정성이 높음

II. 사용처: 서버

III. 구분: Distributed system O, Parallel system과 구분 필요

11. CPU가 임무 끝났는지 아는 방법

A. Polling: 계속 물어보기

I. 특징: 해야 할 것을 못하고 묻기에 집중해야 하기 때문에 효율이 매우 떨어진다

B. Interrupt: 끝났다는 사실을 알리는 신호를 CPU에 보내는 것

I. 특징: CPU가 Interrupt 신호를 받으면 하던 것을 즉각 멈추고, 요청 핸들링을 먼저 해 줌

II. 효율: CPU 단독으로 하면 순차적으로 계속 신호를 받아야 해서 비효율적

III. 해결법:
- detail한 각각의 신호 처리는 I/O controller가, CPU는 큼지막한 명령만 수행
- I/O controller는 이 역할을 수행하기 위해 IR, DR 등의 레지스터를 가지고 있음

12. Synchronous/Asynchronous

A. Synchronous 방식: 입력을 받는 동안 아래 code는 실행되지 않는(blocking) 것

B. Asynchronous 방식: I/O와 다른 작업을 동시에 처리하는 것

I. 예시: 게임을 하면서 백그라운드에서 기타 파일 설치

13. 더 넓은 의미에서의 Interrupt

A. Interrupt: 타이머, 키보드 등의HW 장치에서 I/O 처리가 완료되면 생성되는 신호

B. Trap: 애플리케이션에 의한 것

I. 필요성: HW 운영 전권을 OS가 가지고 있기 때문에 애플리케이션도 I/O 처리를 위해서는 system call의 형태로 OS에 요청을 해야 함

II. 작동 방식: 애플리케이션은 잠시 정지하고, OS 실행 후 다시 어플리케이션이 실행

C. Fault: 오류에 의해서 생기는 경우

I. 예시: 0으로 나눗셈을 하거나, CPU가 자신에게 거는 protection fault 등

D. 분류

I. Hardware interrupt: Interrupt, Fault

II. Software interrupt: Trap

14. DMA(Direct Memory Access)

A. 역할: disk의 한 유닛이 memory로 갈 때 CPU가 계속 관여를 하는 것 막아 효율 증가, 병목현상 완화

B. 방법:

I. DMA controller가 전체 유닛을 옮겨도 될지 허락 요청을 받으면 CPU에 이를 요청

II. 요청이 허락되면 유닛이 디스크에서 메모리로 이동

III. Interrupt 발생

15. Storage hierarchy

A. Memory 종류

I. ROM(read only memory)

- 한번 구우면 수정이 불가
- 옛날 Bios에 사용

II. RAM(random access memory)

- 어디에 접근해도 접근 시간이 동일하고, 수정가능
- 휘발성이기 때문에 전원이 끊기면 다 날라감
- 종류:
  - SRAM(static): 전력 공급 중에 data가 뭉게지지 않기 때문에 전원만 공급해주면 됨
  - DRAM(Dynamic): 그대로 두면 data가 뭉게지기 때문에 신호를 주기적으로 증폭시켜주는 HW 필요
    - 속도: SRAM > DRAM(DRAM은 증폭을 하는 동안에는 data에 접근할 수 없음)
    - 가격: SRAM > DRAM

B. 캐싱

I. 사용 이유: 컴퓨터 전체 성능에 영향을 가장 크게 주는 것은 가장 느린 요소
그래서 제일 느린 요소가 많이 쓰이지 않게 하는 것이 중요

II. 개념: 프로그램을 처음 실행했을 때는 코드 data가 없지만 두번째 실행시에는 data가 남겨져 있어
SSD에 접근하지 않고 곧이어 실행 가능

III. 적용법: register과 cache에만 SRAM을 쓰고, 캐싱을 이용해서 DRAM을 쓰는 SSD 등을 많이 쓰이지 않게 해서 성능 저하를 막자

IV. 기본 가정: "가장 최근에 받은 파일은 가장 가까운 미래에 다시 사용될 확률이 높다"

16. Cache management policy

A. 뜻: 캐싱으로 Register, Cache에서 수행을 많이 한 후, Main memory에 덮어쓰는 정책

B. 종류

I. Write-through: Cache 수정시 memory를 즉각 수정

- 신뢰성 높, 성능 낮

II. Write-back: 쉴 때 memory를 수정

- 신뢰성 낮, 성능 높

17. HDD, SSD

A. HDD

I. 특징: random access가 불가능하며 속도가 느림

II. 원인: data를 찾기 위해서 arm을 해당 위치로 옮겨야 하는데, arm의 속도는 느려서 arm seek time이 길어짐

B. SSD

I. 특징: random access가 가능하며 속도가 빠름

II. USB와 비교: USB는 수정 시에 바뀐 부분을 다시 쓰고 기존 부분을 무효화 시킨다. 이렇게 한 cell 전체가 무효화되면 그때만 clear이 가능한데, cell을 clear할 수 있는 횟수에는 제한이 있다. cell 하나만 망가지면 전체를 다 못쓰게 되기 때문에 USB의 수명이 그렇게 길지 않은 것이다. 반면 SSD에는 수명을 늘리기 위해 clear가 골고루 되게 해주는 핵심 기술인 "wear leveling" 기술이 들어가 있다.

18. Hardware protection

A. CPU protection

I. Timer: 주기적으로 interrupt를 발생시켜서 애플리케이션 다운, 커널을 올려서 CPU 보호

B. Memory protection

I. Protection fault: 어플이 커널의 메모리를 침범하여 write 방지하기 위해 어플을 kill(강제종료)시키는 것

C. I/O protection

I. Dual-mode operation: user, kernel mode 간에 왔다 갔다 하면서 실행됨

- 역할: user mode에서는 I/O에 접근 못하게 하고, kernel mode를 통해서만 접근할 수 있게 하여 I/O 보호
- kernel mode 실행 조건:
  - Bootstrapping(부팅)시
  - 어플리케이션에서 system call(trap) 발생시
  - I/O, HW에서 Interrupt 보낼 때

19. 1st~4th Generation of computers

A. 1세대

I. 시스템: 폰 노이만의 애니악

II. 특징: OS 뿐만 아니라 Assembly Language까지도 없던 시절

B. 2세대

I. 시스템: Batch system

II. 특징:

- 집적도가 높은 트랜지스터를 사용
- 한번에 하나의 프로그램만 동작할 수 있어 I/O 처리시간이 길면 CPU가 놀게 되는 문제

C. 3세대

I. 시스템: Multiprogramming, Time-sharing system

II. 특징:

- 멀티태스킹, 즉 I/O 요청 시에도 실시간으로 CPU의 자원을 다른 어플에 할당할 수 있는 시스템
- 한 어플이 CPU를 독점하게 되는 문제

D. 4세대

I. 특징:

- 컴퓨터 성능 증가, 크기 감소
- PC 등의 형태로 더욱 발전

20. Computing Environments

A. Traditional computing

I. Mainframe system(OS 존재 이후 모든 system 통칭)

B. Mobile computing

I. Hand-held system(메모리 등은 작지만 휴대성이 매우 뛰어남)

C. Real-time embedded computing

I. 특징: 주로 embedded로 구현

II. 종류

- Hard real-time: task의 deadline을 충족 못하면 안됨
- Soft real-time: deadline에 대한 권고만 있을 뿐, 충족 못해도 괜찮음

D. Client-server computing

I. 클라우드, Parallel + Distributed + Storage

E. Cloud computing

I. 특징: 클라우드를 통해 사용자에게 컴퓨팅 서비스를 제공

II. 종류:

- IaaS(Infrastructure): 하드웨어를 제공
- PaaS(Platform): 서버나 DB 등 기반을 제공
- SaaS(Service): 구글 드라이브처럼, 이것에 돌아가는 어플리케이션까지 제공

F. Virtualization

I. 특징: 1개의 HW system 위에 여러 개의 커널이 동작

II. 종류:

- VMM: HW을 통해 구현, 대표적으로 JVM(Java virtual machine)- Platform Independent Execution
- Hypervisor: SW을 통해 구현

21. POSIX

A. 배경: 오픈소스 판매입장에서는 기술 유출 때문에 오픈소스를 할 수 없음, 그래서 binary code로 배포해야 했는데, 그 당시에는 HW 규격도 서로 다르고, system call도 다 달라 Unix 기반이어도 서로 호환 안됨

B. 장점: 통일된 규격 덕분에 Unix 기반 모든 OS는 호환성 100% 보장

22. Linux의 ping-pong 전략

A. 리눅스의 버전은 보통 2.6.0, 3.2.0 등으로 표기되는데, 여기서 마지막 부분인 Minor revision에 주목할 필요가 있다. 일반적으로 마지막 부분이 짝수인 버전은 안정화 버전, 홀수인 버전은 실험적인 버전이어서 이를 참고하여 최적의 버전을 고르면 더 좋을 것이다.

23. Real time과 VxWorks

A. VxWorks

I. 성능이 매우 우수

II. Hard real-time embedded system 중에서 점유율이 매우 높음

III. 가격 매우 비쌈

B. Embedded Linux

I. 오픈소스

II. 현실적으로 사용해볼 수 있는 기술