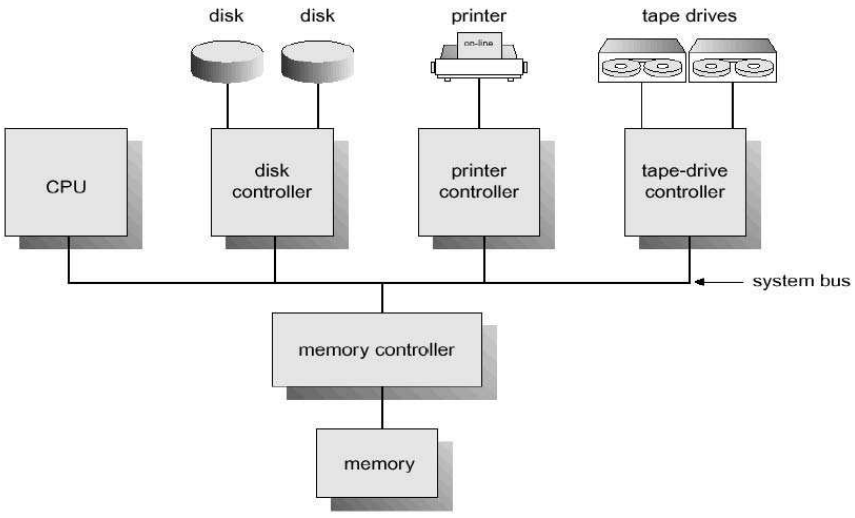
Slide 0

1부 개요
(2장. 컴퓨터 시스템 구조)

컴퓨터 시스템의 동작

Slide 1

- 컴퓨터 시스템



컴퓨터 시스템의 동작 (Cont'd)

Slide 2

- 컴퓨터의 기동
  - 전원인가 혹은 리셋
  - ROM에 있는 부트스트랩 (bootstrap) 프로그램 구동
    - 모든 하드웨어 초기화
    - 운영체제의 커널을 주기억 장치에 적재후 커널 실행 (부팅이라함)
    - 운영체제는 첫 프로세스를 실행 (Unix 에서는 init process)
    - 필요에따라 첫 프로세스가 나머지 프로세스를 생성
    - 사건 (event)의 발생을 기다림

    ☛ PC 환경에서는 BIOS 라고 불림

컴퓨터 시스템의 동작 (Cont'd)

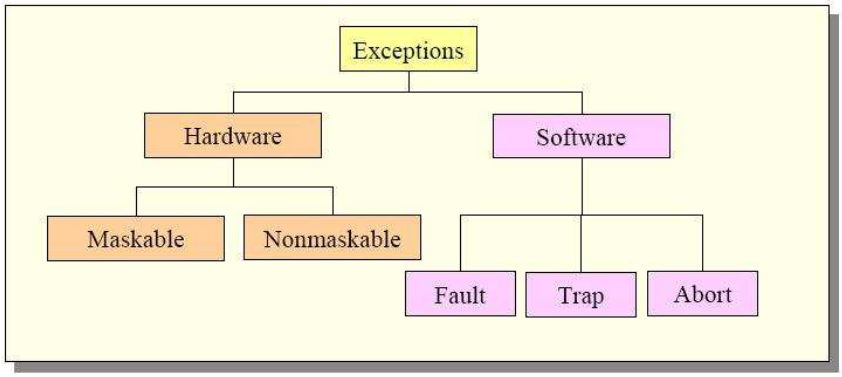
Slide 3

  - 사건의 발생 및 처리
    - 사건의 발생
      - 하드웨어에서 인터럽트가 발생
      - 소프트웨어에서 트랩(trap)이 발생 할 경우
    - 하드웨어 인터럽트의 예): 키보드 입력, 프린터 출력 완료등등
    - 소프트웨어 트랩 발생의 예
      - 커널의 특정 기능을 수행하려고 일부러 발생하는 시스템 호출 (system call) →PC 에서는 소프트웨어 인트럽트라고 함
      - 프로그램 수행 중 문제가 발생된 경우: 예로 0으로 나누기, 권한 위반, 불법 명령등
    - 처리 (운영체제가 처리함)
      1. 현재 작업을 멈추고, 현재 상태를 보관 (주로 인터럽트 스택에)
      2. 인트럽트 종류 분석
      3. 특정 인터럽트 수행 (인터럽트 벡터 활용)
      4. 저장된 상태로 돌아가 원래의 작업을 계속 수행

컴퓨터 시스템의 동작 (Cont'd)

Slide 4

- 예외상황 분류
  - 정상적인 프로그램 수행을 벗어나는 상황
  - exception handler: 예외 상황을 처리하기 위해 미리 정해놓은 루틴



컴퓨터 시스템의 동작 (Cont'd)

Slide 5

- 예외 상황 예

Vector	Description	Error Code	Class
0	DIVIDE ERROR	NO	Fault
1	DEBUG	NO	Fault
3	INT3	NO	Trap
4	OVERFLOW	NO	Trap
5	BOUND	NO	Trap
6	INVALID OP	NO	Fault
7	DEVICE NOT AVAIL	NO	Fault
8	DOUBLE FAULT	YES(항상 0)	Fault
9	SEGMENT OVERRUN	NO	Abort
10	INVALID TSS	YES	Abort
11	SEGMENT NOT PRESENT	YES	Fault
12	STACK EXECPTION	YES	Fault
13	GENERAL PROTECTION	YES	Fault
14	PAGE FAULT	NO	Trap
16	COPROCESSOR ERROR		Fault

컴퓨터 시스템의 동작 (Cont'd)

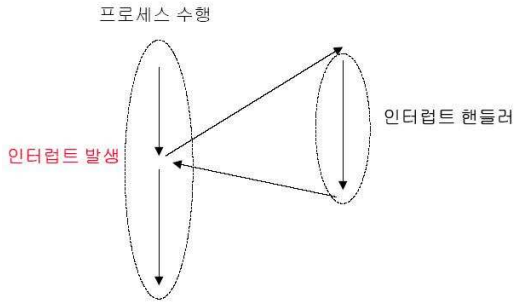
Slide 6

- 예외 처리 벡터
  - 모든 exception은 고유한 vector(8bit)를 가지고 있음
  - 하나의 vector가 하나의 handler를 가질 수 있음
  - Software exception vector는 고정되어 있음
  - Hardware interrupt vector는 수동적으로(own) 할당
- 예외 처리 과정
  - CPU는 exception의 소스(Vector)를 알아냄
  - Vector를 이용하여 IDT에 있는 ISR (exception handler)를 호출
    - IDT (Interrupt Descriptor Table)
    - ISR (Interrupt Service Routine)

컴퓨터 시스템의 동작 (Cont'd)

Slide 7

- 인터럽트
  - 인터럽트(interrupt)는 제어를 인터럽트 서비스 루틴(interrupt service routine)으로 이동시킴
  - 인터럽트 벡터(interrupt vector)를 통해서 인터럽트 서비스 루틴의 주소를 찾음
  - 처리 (interrupt handling)
    - 현재 수행중인 작업을 중지하고
    - 프로세서 정보를 저장한 후
    - 인터럽트 서비스 실행하고
    - 서비스가 종료되면
    - 수행 중이던 프로그램 복귀

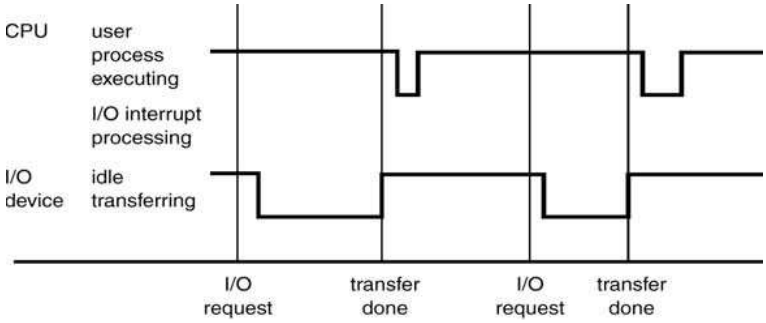


  - 인터럽트는 MASK 할 수 있는 것과 그렇지 않은 것이 있음 (NMI)
  - 트랩(trap)은 에러 또는 사용자의 요구에 따라서 소프트웨어적으로 발생되는 인터럽트임

컴퓨터 시스템의 동작 (Cont'd)

Slide 8

- 인터럽트 처리

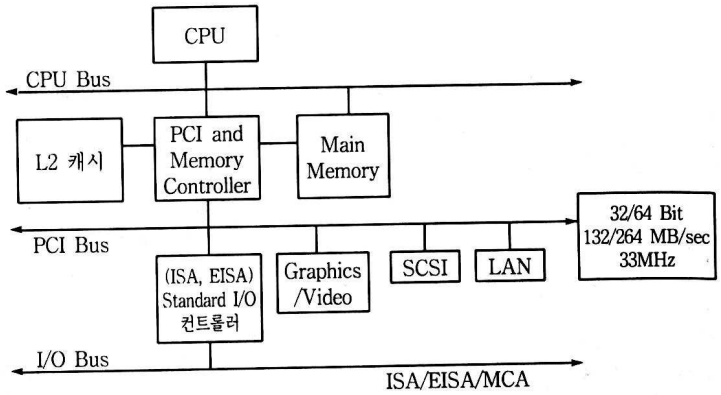


입출력 구조

Slide 9

- 입출력 장치의 연결

CPU	시스템 버스	장치 제어기	장치
PIII	PCI	SCSI controller	SCSI HDD



[그림 19-5] PCI 버스 구조

입출력 구조 (Cont'd)

Slide 10

- 입출력 동작
  - 입출력과 CPU 의 실행은 동시에 진행될 수 있음
  - 장치제어기는 해당 장치의 입출력 및 제어를 책임
  - 장치제어기는 데이터 저장을 위하여 로컬 버퍼를 갖고 있음
  - CPU 는 메모리 와 장치제어기의 버퍼사이에 데이터 이동을 할 수 있음
  - 장치제어기는 장치와 장치제어기 버퍼사이에 입출력을 수행함
  - 장치제어기는 인터럽트를 이용하여 CPU 에게 입출력의 완료를 알림
    →(인터럽트를 제공하지 않는 장치도 있음)
  - 장치제어기 (device controller)를 제어하는 프로그램
    →장치 드라이버 (device driver)

입출력 구조 (Cont'd)

Slide 11

- 입출력 순서
  - 프로세스가 입출력을 요청 (프로그램의 입출력 수행 부분이 실행되는 것임)
  - 운영체제가 요청을 받아들여서 장치드라이버에게 입출력을 요청
  - 장치드라이버는 장치제어기 레지스터 중 작업 요청 레지스터에 입출력 요청
    →(장치제어기의 특정 레지스터에 특정 명령을 써 넣음)
  - 입출력이 일어남
    * 장치제어기는 장치에게 입출력을 요청
    * 장치는 요청에 따른 작업을 수행하여 데이터(입력의 경우)나 결과(출력의 경우)를 장치제어기에 알려줌
    * 장치제어기는 버퍼에 이를 저장
    * 모든 데이터 처리가 끝날때까지 반복
  - 장치제어기는 입출력의 완료를 알림
  - 장치드라이버는 입출력의 완료를 운영체제에게 알림
  - 운영체제는 프로세스에게 입출력의 완료를 알림

입출력 구조 (Cont'd)

Slide 12

- 입출력 방법 (하드웨어 특성에 따른 구분)
  - 폴링 방법
    - 배경
      - 인터럽트를 제공하지 않는 장치로부터의 입출력 방법
      - 장치제어기는 보통 인터럽트를 제공
      - 그러므로 대부분 장치제어기를 사용하지 않는 단순한 장치에서 사용하는 방법
    - 방법
      - 장치드라이버는 장치 (혹은 제어기) 레지스터 중 작업 요청 레지스터에 입출력 요청
      - 해당 장치 (혹은 제어기)가 입출력을 수행
      - 장치 드라이버는 레지스터의 작업완료 표시를 보고 입출력 완료됨을 확인
    - 문제점
      - CPU 가 입출력이 완료될때까지 기다리면서 장치 (혹은 제어기) 상태를 검사해야함
      - 다른 작업을 할 수 없음

입출력 구조 (Cont'd)

Slide 13

  - 인터럽트 방법
    - 배경
      - 인터럽트를 제공하는 장치제어기를 사용하는 장치와의 입출력시 사용
      - 단순한 기능이 아닌 많은 기능을 제공하는 장치에서 사용
    - 방법
      - 장치드라이버는 장치 제어기 레지스터 중 작업 요청 레지스터에 입출력 요청
      - 장치 제어기는 장치에게 입출력을 요청
        - ➥ 입력일 경우 장치에서 들어온 데이터를 장치제어기 버퍼에 저장함
        - ➥ 출력일 경우 장치제어기 버퍼에 있는 데이터를 장치로 보냄
        - ➥ 입출력이 완료되면 장치 제어기는 CPU 로 인터럽트를 발생
      - 인터럽트 서비스 루틴에서 입출력 완료를 알림
      - 장치드라이버가 입출력 작업의 완료를 인지

입출력 구조 (Cont'd)

Slide 14

- 입출력 방법 (동작 방법에 따른 구분)
  - 동기식 (synchronous) 방법
    - 입출력이 완료될때까지 기다리는 방법
      - 기다리는 방법은 특수한 대기 (wait) 명령을 사용하던지
      - 이러한 명령이 없는 CPU의 경우 대기 루푸 (wait loop)를 사용
    - 장점
      - 입출력 인터럽트가 발생하면 어떤 입출력 요구에 의하여 어떤 장치에서 발생한 것인지를 쉽게 알 수 있음
    - 문제점
      - 한 순간에 하나의 입출력만 가능 (여러장치가 동시에 수행될 수 있음에도 불구하고)
      - 입출력 중에는 대기함으로 다른 입출력뿐만 아니라 다른 연산도 불가능

입출력 구조 (Cont'd)

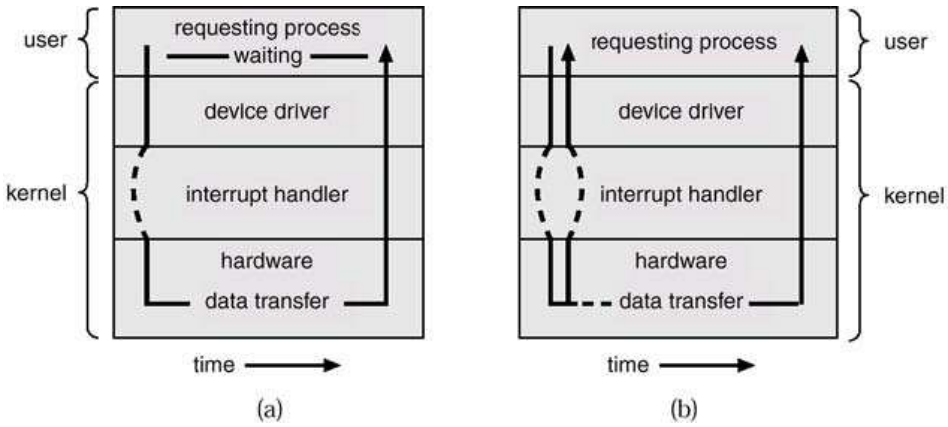
Slide 15

  - 비동기식 (asynchronous) 방법
    - 입출력이 완성되지 않았으나 바로 해당 프로세스로 리턴하는 방법
    - 장점
      - 해당 프로세스가 계속실행되므로 입출력 중에 다른 연산이 가능
      - 다른 장치로의 입출력 요구가 있을 경우 이를 처리하여 동시에 다수의 입출력이 가능
    - 부가적인 작업
      - 해당 프로세스가 대기 입출력의 완료를 알 수 있도록하는 부가적인 시스템 호출이 필요함
      - 입출력 요구가 계속발생할 수 있음으로 장치별 입출력 요구 테이블을 만들어 관리해야함

입출력 구조 (Cont'd)

– 동기식 비동기식 방법

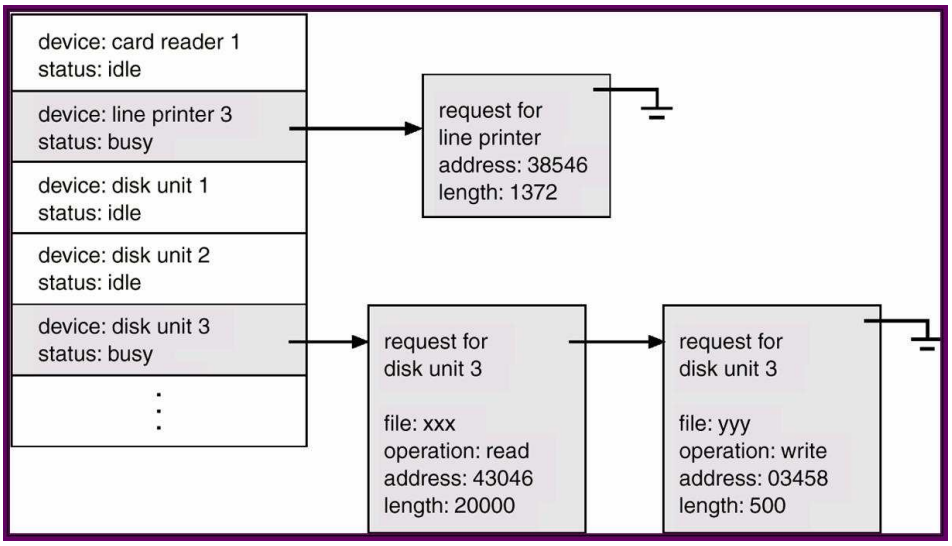
Slide 16



입출력 구조 (Cont'd)

– 장치-상태 테이블

Slide 17



DMA 구조

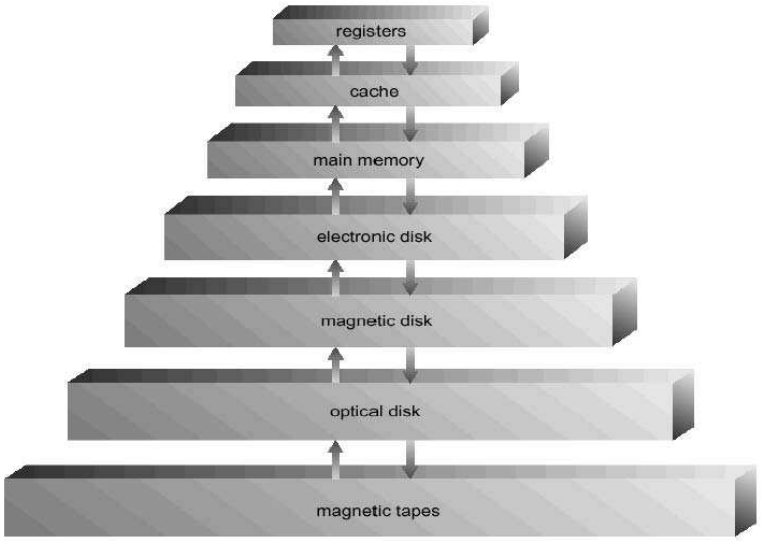
Slide 18

- DMA (Direct Memory Access) 란?
  - 고속의 장치와의 입출력을 지원하는 장치
  - CPU 의 간섭 없이 {메모리 혹은 IO} ↔{메모리 혹은 IO} 사이에 데이터 전송
  - 블록별로 인터럽트가 발생
- 장점
  - CPU 보다 고속의 데이터 전송이 가능
  - DMA 가 데이터 전송시 CPU 는 다른 작업 가능
  - CPU 가 메모리 접속이 필요한 경우 DMA 가 작업을 종료한 후에 접속해야함
    →캐쉬 히트가 나면 캐쉬에서 가져와서 작업할 수 있음 →성능 증가

저장장치 구조

Slide 19

- 구성



저장장치 구조 (Cont'd)

Slide 20

- 주기억장치 (Main Memory)
  - CPU 는 CPU 내부의 레지스터와 주기억장치만을 직접 액세스 가능함
  - 프로그램이 수행되기 위해서는 주기억장치에 적재되어야 함
  - 속도측면에서의 한계
    - CPU의 속도보다 느림
  - 해결
    - 캐시 (cache) 메모리 사용
    - 고속의 메모리로 보통 SRAM 사용
    - 용량은 메모리보다 작음
    - 자주 사용하는 것을 캐시에 남겨 놓음

저장장치 구조 (Cont'd)

Slide 21

  - 용량 및 휘발성의 한계
    - 용량이 충분히 크지 못하다.
    - 휘발성이다.
  - 해결
    - 보조기억장치가 필요함
    - 비휘발성이어야함 (전원이 꺼져도 정보가 남아있어야함)
    - 용량이 대용량이어야함 (대부분의 프로그램을 저장하고 있어야함)
  - 보조 저장장치
    - 주로 자기디스크를 사용 (하드디스크)
    - 기타 자기테이프, CD-ROM 등이 있음

저장장치 구조 (Cont'd)

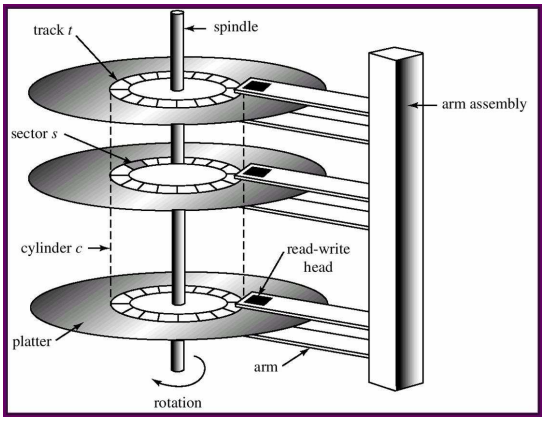
Slide 22

- 주기억장치를 위한 운영체제의 기능
  - 주기억 장치의 어떤 부분이 사용되고 있는지 추적할 수 있어야 함
  - 필요에 따라서 메모리를 할당 및 해제 할 수 있어야 함
  - 다른 프로세스의 접근으로부터 보호해야 함
- Memory-mapped I/O (I/O mapped I/O)
  - 메모리 주소상에 I/O 장치가 매핑되어 있음
  - 메모리와의 데이터 교환하는 방법으로 입출력함
  - 고속의 장치를 위해 사용
  - IBM PC 에서 주 메모리의 일부분을 비디오메모리로 사용했었음

저장장치 구조 (Cont'd)

Slide 23

- 자기 디스크
  - 플래터(platter): 원판
  - 트랙 (track): 회전축을중심으로 데이터가 기록되는 동심원
  - 섹터 (sector): 데이터가 저장되는 단위로 트랙을 여러 개로 나눈 것
  - 실린더 (cylinder): 같은 직경을 갖는 트랙들의 집합



저장장치 구조 (Cont'd)

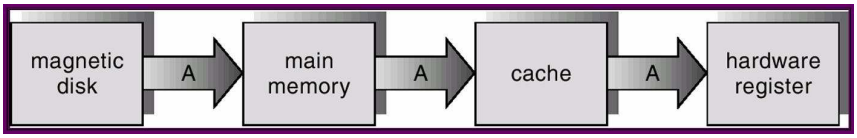
Slide 24

- 접근시간
  - 탐색시간 (seek time): 헤드가 원하는 실린더로 움직이는데 걸리는 시간
  - 회전지연 (rotation latency): 헤드가 원하는 섹터로 회전하는데 걸리는 시간
- I/O 버스 연결
  - EIDE (Enhanced Integrated Drive Electronics)
  - ATA (Advanced Technology Attachment)
  - SCSI (Small Computer Systems Interface)
- 자기 테이프
  - SAM (Sequential Access Memory)
  - 대용량으로 주로 백업 및 자주사용하지 않는 정보의 저장용으로 사용

저장장치 구조 (Cont'd)

Slide 25

- Coherency 와 consistency
  - 데이터를 읽어 옮



  - 데이터의 변경으로 인한 문제
    - 단일 프로세스: 최상위 메모리만 변경하면 됨
    - 다중 프로세스: 모든 프로세스가 가장 최근 수정된 값을 액세스 해야함
    - 다중 프로세서: 각 프로세서 캐쉬에 복사본이 존재 →하나의 캐쉬에서 수정되면 모든 다른 캐쉬도 수정되어야함
    - 분산 환경: 파일수준에서 일관성이 유지되어야함

하드웨어 보호

Slide 26

- 배경
  - 자원의 효과적인 활용을 위해 공유기법을 사용
    - 멀티 프로세스: 여러 프로세스가 동시에 수행됨
    - 스풀링: 프로세스와 입출력이 동시에 수행됨
  - 오류 발생시 다른 프로세스에 영향을 줌
    - 불법적인 명령
    - 운영체제 영역의 메모리 접근등
  - 처리
    - 오류가 발생하면
    - 해당 프로세스를 강제 종료하고
    - 트랩을 발생시켜 인터럽트 벡터를 통해 운영체제로 제어권을 넘김
    - 운영체제는 오류 메세지를 출력하고 프로그램의 내용을 덤프함
  - 다른 프로세스를 보호하지 못하는 운영체제
    - MS-DOS
    - 매킨토시 운영체제

하드웨어 보호 (Cont'd)

Slide 27

- 하드웨어 보호의 종류
  - 이중모드 동작
  - 입출력보호 (I/O Protection)
  - 메모리 보호 (Memory Protection)
  - CPU 보호
- 이중모드 동작 (Dual-Mode Operation)
  - CPU의 동작시 두 모드로 동작하게 함
    - 사용자 모드
    - 모니터 모드 (슈퍼바이저 모드, 시스템 모드, 특권 모드로도 불림)
    - CPU 의 레지스터내에 한 비트로 구분 (보통 PSW에 있음)
  - 모드 전환
    - 초기 부팅시 모니터 모드로 수행
    - 운영체제 수행시 모니터 모드로 수행
    - 사용자 프로그램 수행시 사용자 모드로 전환됨
    - 사용자 프로그램 수행 중 트랩이나 인터럽트 발생하면 모니터모드로 수행 전환

하드웨어 보호 (Cont'd)

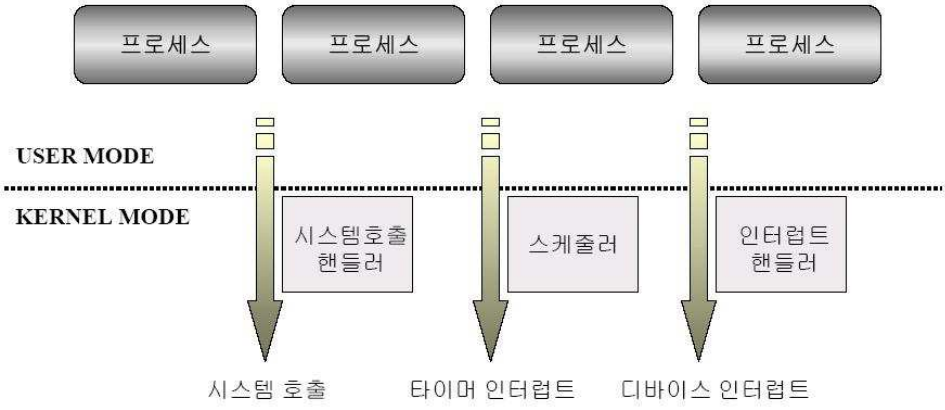
Slide 28

- 하드웨어 보호
  - 사용자 모드로 수행시 운영체제만 수행하는 특권명령을 실행하면
  - 이의 실행을 막고
  - 불법명령어 트랩 발생
- 8088 의 경우
  - 모드비트가 없음
  - 사용자 프로그램이 MS-DOS 운영체제를 지워거나 덮어써서 동작 불가하게 할 수 있음
- 사용자 프로그램에서 특권명령 수행이 필요한 경우
  - 시스템 호출을 통하여 운영체제에게 의뢰
  - 하드웨어는 이를 소프트웨어 인터럽트로 간주

하드웨어 보호 (Cont'd)

Slide 29

- 사용자/모니터 모드의 전환



하드웨어 보호 (Cont'd)

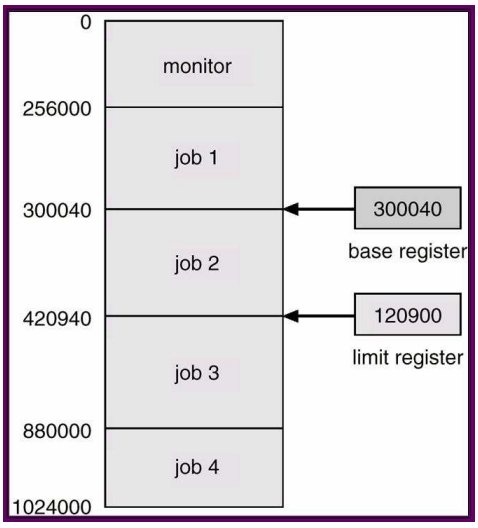
Slide 30

- 입출력보호 (I/O Protection)
  - 모든 입출력 명령을 특권명령으로 분류함
  - 사용자 프로그램에서 바로 입출력 할 수 없음
  - 시스템 호출을 통하여 입출력
- 메모리 보호 (Memory Protection)
  - 보호해야할 대상
    - 인터럽트 벡터 테이블
    - 인터럽트 서비스 루틴
    - 사용자 프로세스와 운영체제
    - 프로세스사이의 데이터

하드웨어 보호 (Cont'd)

Slide 31

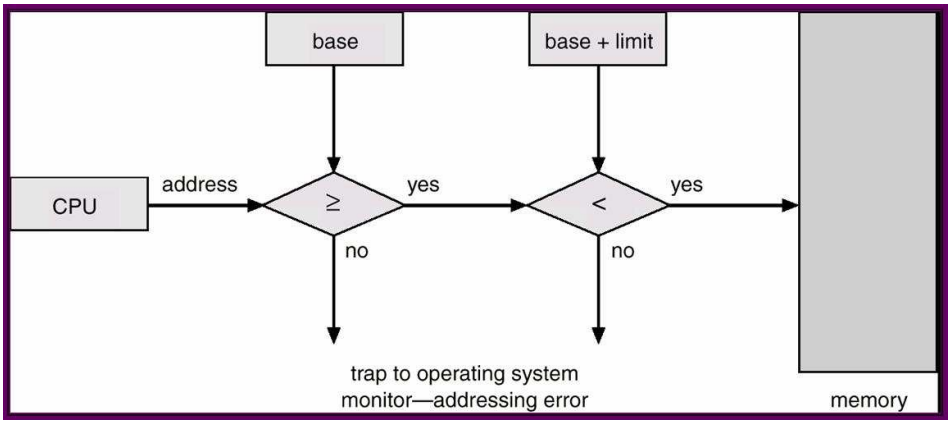
- 보호 방법
  - 기준 레지스터 와 한계 레지스터를 사용
  - 기준 레지스터 ~기준 레지스터 + 한계 레지스터 만 액세스 가능
  - 이외에 액세스는 트랩을 발생시킴
  - 기준 레지스터와 한계 레지스터의 설정은 특권 명령으로만 가능
  - 특권명령은 운영체제만 실행 가능



하드웨어 보호 (Cont'd)

Slide 32

– 운영체제의 메모리 보호



하드웨어 보호 (Cont'd)

Slide 33

- CPU 보호
  - 내용
    - 사용자 프로그램 오류나 무한루프로 부터 운영체제가 제어권을 확보할 수 있도록 보호
    - 타이머를 사용
  - 보호 방법
    - 사용자 프로그램 실행 전에 타이머 값을 설정
    - 매 클럭에서 타이머 값을 1씩 감소
    - 타이머 값이 0 이되면 인터럽트 발생
    - 인터럽트 처리로 운영체제가 제어권 확보
    - 타이머 값 설정은 특권명령
  - 시분할 (time sharing) 구현
    - 타이머의 보다 일반적인 사용
    - 타이머를 사용하여 프로세스별로 특정 시간동안 CPU를 할당함
    - 인터럽트가 걸리면 운영체제는 새로운 프로세스를 수행 시킴 →context switch 라함