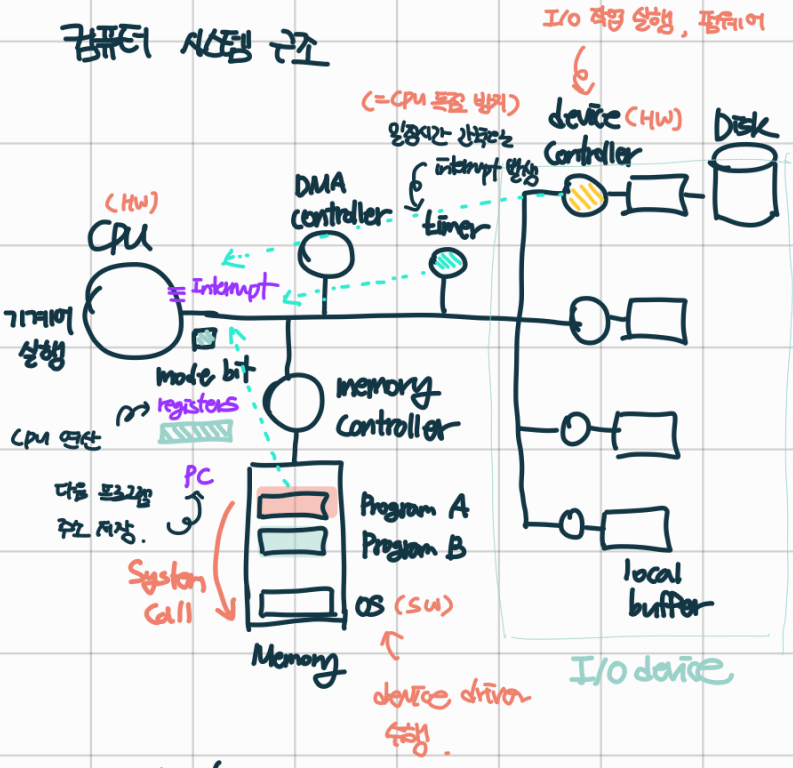
운영체제의 분류

- 동시작업 여부
  - 단일작업 (Single tasking)
    : 한 번에 하나의 작업만 처리 ex) MS-DOS
  - 다중작업 (multi tasking)
    : 동시에 두 개 이상의 작업 처리 ex) UNIX, Window
- 사용자의 수
  - 단일 사용자 (Single user)
    ex) MS-DOS, Window
  - 다중 사용자 (multi user)
    ex) UNIX, NT server
- 처리방식
  - 일괄처리 (batch processing)
    · 작업 요청의 일정량 모아서 한꺼번에 처리
    · 작업이 완전 끝날 때까지 기다려야 함
    ex) 초기 Push Card 처리 시스템
  - 시분할 (time sharing) ← 현재 사용
    · 여러 작업을 수행할 때 컴퓨터 처리 능력을 일정한 시간 단위로 분할 → CPU
    · 일괄처리 시스템에 비해 짧은 응답 시간을 가짐
    ex) UNIX
    · interactive한 방식
  - 실시간 (Realtime Os)
    · Deadline 존재 (반드시 해당시간안에 종료되어야 함)
    ex) 반도체 장비, 미사일 제어
    · 실시간 시스템의 2개로 확장
      - hard realtime system
      - soft realtime system

- 유닉스 (UNIX) ⇒ 현재 운영
  - C언어
  - 높은 이식성
  - 최소한의 커널 구조
  - 복잡한 시스템에 맞게 확장 용이 → 왜지?
  - 소스코드 공개
  - 프로그램 개발에 용이
- DOS (Disk Operating System)
  - 단일 사용자용 OS, 메모리 관리 능력의 한계
- MS Windows
  - 다중 작업과 GUI 기반 OS
  - Plug and Play, 네트워크 환경 강화
  - DOS용 응용 프로그램과 호환성
  - 불안정성
  - 풍부한 자료 SW

컴퓨터 시스템 구조

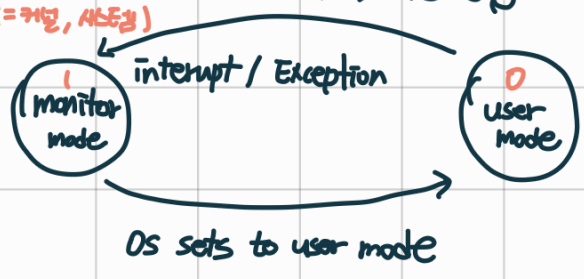


- mode bit

: 사용자 프로그램의 잘못된 수행으로 다른 프로그램 및 OS에 피해가지 않도록 보호 장치 필요

- 1: 사용자 모드 : 사용자 프로그램 수행 → 제한된 기계어 실행
- 0: 모니터 모드 : OS 코드 수행, 특권 명령

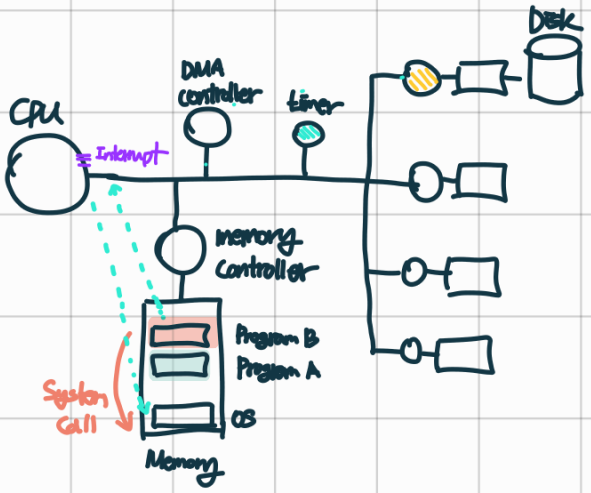
↳ (=커널, 시스템)



*시스템 콜

: 사용자 프로그램이 운영체제의 서비스를 받기 위해 커널 함수 호출하는 것

ex) 운영체제에게 CPU 주도권을 넘기는 것



Interrupt

: 인터럽트를 당한 시점의 레지스터와 Program counter를 save 한 후 CPU의 제어를 인터럽트 처리 루틴에 넘기다.

- Interrupt (하드웨어 인터럽트)
  - : 하드웨어가 발생시킨 인터럽트
- Trap (소프트웨어 인터럽트)
  - · exception : 프로그램이 오류를 범함
  - · System call : 프로그램이 커널함수를 호출한 경우

Device Controller

- I/O device controller
  - · 해당 I/O 장치유형을 관리하는 일종의 작은 CPU
  - · 제어정보를 위해 control register, status register를 가짐
  - · local buffer를 가짐 (일종의 data register)
- I/O는 실제 device와 local buffer 사이에서 일어남
- device controller는 I/O 작업이 끝났을 경우 interrupt로 CPU에 알림

* device driver (장치 구동기)
  - : OS코드 중 각 장치별 처리기 → SW

* device controller (장치제어기)
  - : 각 장치를 통제하는 일종의 작은 CPU → HW

* 운영체제로 CPU가 넘어오는 경우

① HW 장치들이 Interrupt를 거는 경우

② 프로그램 SW가 직접 인터럽트 라인을 세팅

ex) System call (I/O 작업)

동기식 입출력과 비동기식 입출력

· 동기식 입출력 (Synchronous I/O)

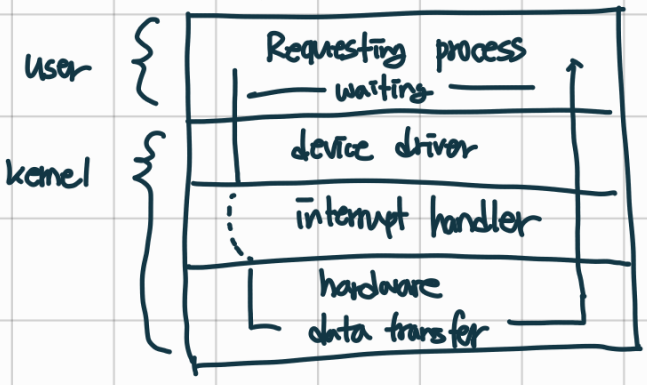
- I/O 요청 후 입출력 작업이 완료된 후에야 제어가 사용자 프로그램에 넘어감
- 구현 방법 ①
  · I/O가 끝날 때까지 CPU를 낭비시킴
  · DMA 사용 하나의 I/O만 일어날 수 있음
- 구현 방법 ②
  · I/O가 완료될 때까지 해당 프로그램에게서 CPU를 빼앗음
  · I/O 처리를 기다리는 줄에 그 프로그램을 줄 세움
  · 다른 프로그램에게 CPU를 줌

· 비동기식 입출력 (asynchronous I/O)

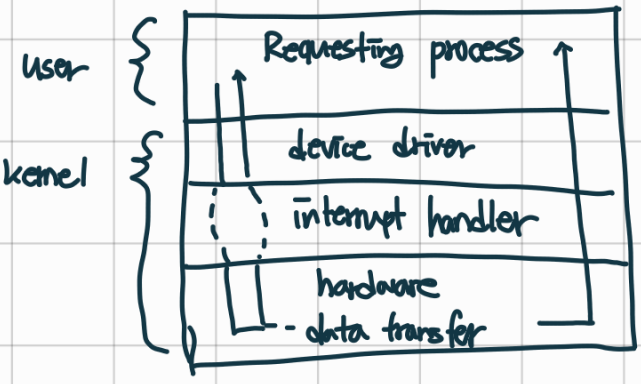
- I/O가 시작된 후 입출력 작업이 끝나기를 기다리지 않고 제어가 사용자 프로그램에 즉시 넘어감

* 두 방법 모두 I/O의 완료는 인터럽트로 알려줌

Synchronous



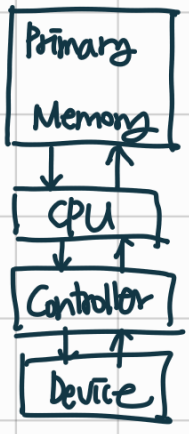
Asynchronous



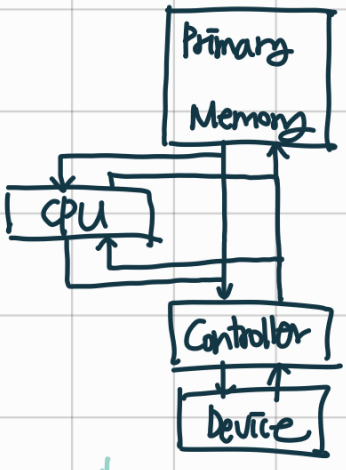
너무 자주 interrupt 걸리는 것을 방지

DMA (Direct Memory Access)

- 빠른 입출력 장치를 메모리에 가까운 속도로 처리하기 위해 사용
- CPU의 중재 없이 device controller가 device의 buffer storage의 내용을 메모리에 block 단위로 직접 전송
- 바이트 단위가 아니라 block 단위로 인터럽트를 발생시킴



→ memory 접근 권한이 CPU 밖에 없다.

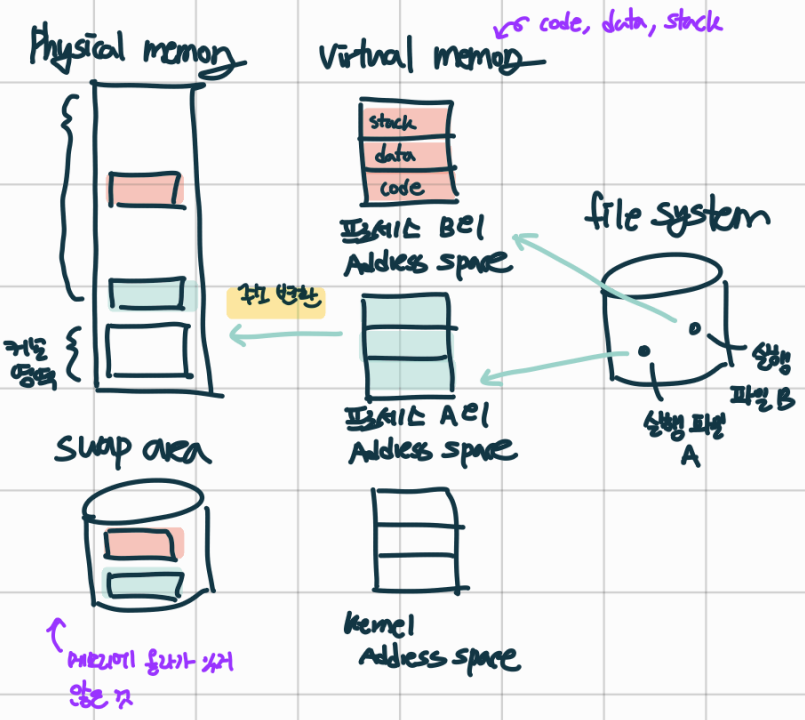


↓
CPU 와이 DMA Controller가 메모리에 직접 접근하도록.
⇒ interrupt 빈도수 ↓

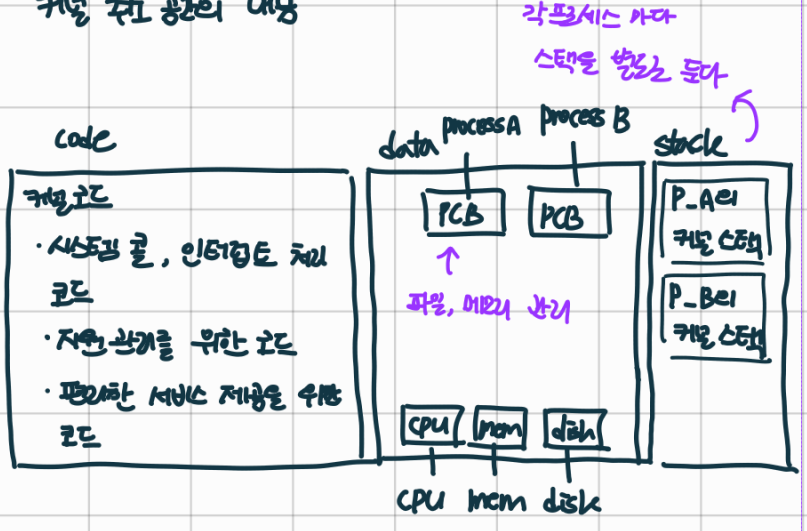
보다 나은 입출력 기법이

- I/O를 수행하는 special instruction에 의해
- Memory Mapped I/O에 의해

프로그램의 실행 (메모리 load)



커널 주소 공간의 내용



* I/O 작업은 특권 명령.

사용자 프로그램이 사용하는 함수

- function
  - 사용자 정의 함수
    - 자신의 프로그램에서 정의한 함수
  - 라이브러리 함수
    - 갖다 쓴 함수
    - 자신의 프로그램의 실행 파일에 포함되어 있다

→ 프로세스 Address space

  - 커널 함수
    - OS 프로그램의 함수
    - 커널 함수와 호출 = 시스템 콜

→ Code → kernel Address space

프로그램의 실행