# 운영체제

## 운영체제💻

## Operating System

### 운영체제 서론

> 운영체제란 무엇인가?

PC를 구입하면 딸려오는것? Windows, Linux, Mac Os, MS DOS.. 등등

> 그렇다면.. 운영체제가 없는 컴퓨터는 어떻게 될까? 컴퓨터 구조로 생각해보자!

컴퓨터라는 것은 이런 구조로 되어있다.

1 Processor(처리기), Main Memory(메모리) 가 있다. 기본적으로 컴퓨터 전원을 키면, 프로세서가 메모리에 있는 명령을 들고와서 실행을 하고, 그다음 명령을 또 들고와서 실행을 하고.. 하는 식으로 작동을 하는데?

2 ⇒ 곧, 메모리에 명령들(Instruction)을 기록해 둬야 하는데? (명령들의 집합이 프로그램) 만약 운영체제가 없다면 메모리에 자기 멋대로의 임의의 값이 들어있게 된다.

3 메모리는 `휘발성` 메모리이기 때문에? 기본적으로 실행 프로그램들은 하드디스크에 들어있게 되는데, 이걸 메모리에 올리는것도 못함 OS가 없으면.

4 현대 컴퓨터는 또, 여러개의 프로그램을 동시에 올려서 실행을 하는데? 프로세서는 하나뿐이잖아? 이런 역할도 OS가 함.

5 뭘 하드디스크에 저장하거나, 프린터를 연결해서 실행시키거나.. 하는것도 다 OS.

---

`정리`

1. 성능향상(Performance) = 컴퓨터의 하드웨어를 잘 관리(프로세서, 메모리, 디스크, 주변장치 등등) + 좋은 OS가 있다면 하드웨어 성능도 더 끌어낼 수 있음.
2. 사용자에게 편의성(Convenience) 제공 = 애들도 몇번 해보면 금방 씀. (예전엔 컴 쓰는게 어려워서 컴퓨터를 사용하는 Operator라는 직업도 있었음)

⇒ 결론 : `컴퓨터 하드웨어를 관리하는 프로그램` Control program for computer

---

> 부팅(Booting)

컴퓨터는 프로세서 + 메인 메모리(주기억장치)가 가장 핵심적인 장치이고, 하드디스크(보조기억장치)를 가지고 있음.

여기서 메인메모리는 2가지로 나뉘는데, 램과 롬임. (근데 메인메모리 대부분은 RAM)

1 RAM(보통 DRAM) → 일반적으로 요즘 작은거라 해봤자 4GB 부터 꼽음

2 ROM(Read Only Memory) → 많아야 수백 kb 정도. 컴퓨터에서 약간의 롬이 필요함. ( EX)휴대폰의 플래시 메모리 )

근데 이러면 롬은 왜필요하나? ⟹ RAM에 있는 내용은 `휘발성` 메모리라 컴퓨터 껐다 키는 순간 죄다 날아감. 롬은 전원을 꺼도 내용이 유지되고 있음.

> 주의 : 하드디스크도 전원과는 관계 없음. 껐다 킨다고 날아가는게 아님. 그래서 ROM이 하드디스크냐? 이건 아님. 카테고리가 이미 'Main Memory' 니까.

폰이나 PC의 전원을 킨다면? 가장 처음으로 프로세서는 (RAM이 아니고) `ROM`안에 있는 Instruction을 읽어와서 실행하게 됨!! ⇒ 다음과 같은 순서

1. POST(Power-On Self-Test)가 제일먼저 실행 = 전기 키면 셀프테스팅 하는것(제대로 환경설정이 되어있는가, 키보드가 꽂혀 있는가, 메모리는 얼마 있는가 등등 체크)
2. Boot Loader 실행 = 컴퓨터는 보통 `하드디스크`안에 OS를 설치해 두는데, 부트로더가 하는 일은 컴퓨터 켜지면 하드디스크를 뒤져서 OS를 메인 메모리(RAM 영역)로 올려줌.

   ⇒ 이렇게 메인 메모리로 올리는 것을 `Boot` 라고 부름.
3. 2단계 까지 했으면 ROM은 자기 역할을 끝냄.

OS가 Main Memory로 올라오면 바탕화면 뜨고, 우리가 익숙한 그 화면이 보이게 됨. 지금은 OS가 메인 메모리에 상주하면서 이제 받는 명령을 처리할 준비가 되었다는 뜻. (컴퓨터 전원을 끄면 OS도 메인 메모리에서 날아감)

조금 헷갈릴수도 있는 부분이, 컴퓨터를 키고 이것저것 프로그램(게임이나 크롬, 한글등)을 실행할때 그걸 꺼버리면 메인 메모리에서도 내려가는데, OS는 컴퓨터 전원이 들어와 있는 동안은 메모리에 상주함. = 그래서 OS를 `Memory Resident` 라고 함.

---

## OS는 사실 두 부분으로 나뉨. Kernel(핵심) & Shell(껍질)

OS가 하드웨어를 감싸서 관리를 해주고 있는 형태라고 생각하면 되는데. 실제로 OS가 하드웨어를 제어하고 관리하는 부분을 `OS의 커널, Kernel(핵심, 핵)` 이라고 부름.

가령, 프로그램을 실제로 실행하고 싶은 경우 (윈도우즈의 경우 유저 인터페이스는 우리가 익숙한 바탕화면 그거임) 아이콘에 마우스 화살표 갖다 대고 더블클릭해서 실행 하는데? 이건 내가 어떤 프로그램을 실행하라! 라고 명령을 내린거임. 이런 명령을 내릴 수 있도록 만들어 주는 것을 운영체제의 껍데기 부분, `Command Interpreter` 라고 부름. (리눅스는 기본적으로 CLI 라서 $ (달러사인) 오른쪽에 ls 이런거 치거나 함 + who 치면 어떤 유저가 쓰는지, df(disk free)치면 보조기억장치 전체 용량중 사용 용량 이런거 확인 가능)

리눅스로 예를 들면, 이런 사용자 명령어 who 내리면 Shell이 이게 뭔 명령인지 해석하여 동작해서 OS가 일할 수 있도록 기반을 마련.

`정리`

- `커널` = 실제 관리 프로그램 (Cpu, Memory 등 관리)
- `쉘` = 사용자가 명령 내린걸 해석해서 결과를 화면에 보여주는 껍질

그래서 커널은 관리프로그램이라 눈에 거의 안보이는것.

⟹ 만약 누가 너 리눅스 사용법 아냐? 라고 물었을때 $ 옆에 ls 하면 목록 뜨지 ㅇㅇ, 뭐 그런 측면에서 적당히 사용법 알아~ 라고 한다면 이건 커널을 안다는게 아니라 쉘(사용법)을 안다는거임

```
# 운영체제 내용을 배운다고 하면 그러니까 이 강의에서는 "커널" !!! 을 배우는거임.Application[ OS [ Hardware ]  ]  <= 이런 포함관계
```

일반적으로 어플리케이션은 OS 위에서 실행되니까, OS가 달라지면 앱이 실행이 안됨.

뭔뜻이냐면? 윈도우에서 한글 프로그램 파일 긁어다가 맥에 집어넣으면 이거 안됨. 사실 하드웨어는 뭐 인텔 CPU 써서 같을 수 있는데? OS가 다르니까 안되는거.

`어플리케이션은 하드웨어 위에서 도는게 아니라 OS위에서 도는거임`

컴퓨터 구조 수업에서는 컴퓨터를 CPU 메모리 보조기억장치의 세트로 개념도를 그렸다면, 이 수업에서는 Application → Os → Hardware 식으로 그릴 수 있음.

OS는 결국 정부와 비슷하다고 할 수 있는데, 주어진 자원을 활용하여 일을 하는것. 정부에 여러 부처가 있듯, OS에는 `Processor Management`, `Memory Management`, `IO Management`(프린터 키보드 등 관리), `File Management` , `Network Management`, `Security or Protection Management` 등등 여러가지가 있음. 이중 제일 중요한 부처는 Processor Management이고 그다음은 Memory Management임. 이 두개가 수업의 핵심.

OS의 다른 이름. 하드웨어를 자원 이라고 함!

= `자원관리자` (resource manager) = `자원할당자` (resource allocator)

# 운영체제 역사

## 역사

### 1. No O/S

- 1940년대 말 ( 2차대전 중 개발)
- 거대해서 한 건물안에 설치
    1. 카드리더 : 가장 큰 장치, 입력장치, 천공카드(구멍 뚫린 종이) 입력
    2. 처리기 : processer & memory
    3. 프린터 : 출력장치, output을 line print
- 동작 순서 : card reader -> memory -> proessing -> printer
    1. 프로그램을 천공카드에 표시해 메모리에 적재
    2. 컴파일러를 천공카드에 표시해 메모리에 적재
    3. 컴파일러가 프로그램 번역 -> 기계어
    4. 처리기가 기계어 실행 -> 출력

### 2. Batch processing system (일괄처리) - 최초의 O/S

- 프로그램을 수행할 때마다 컴파일->링크->로딩 순서를 오퍼레이터가 직접 입력함

  => 이러한 과정을 하나의 프로그램으로 작성해 메모리에 안에 할당해 자동화한 것이 batch processing
- resident monitor : 메모리에 상주하며 일련의 일(컴파일, 링크, 로딩 등)을 하는 프로그램

### 3. Multiprogramming system (다중 프로그래밍)

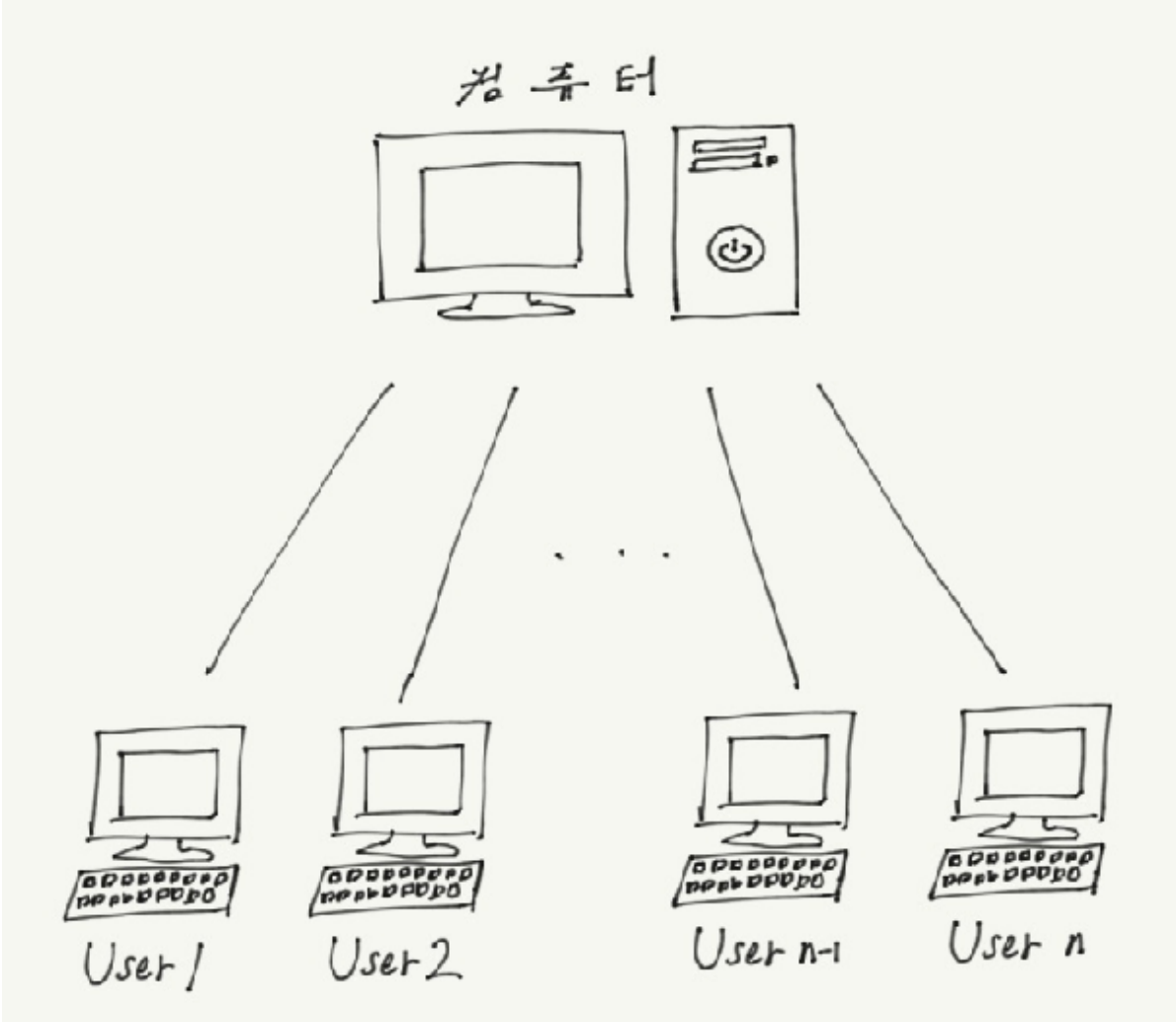
- 1960년대
- 하드웨어의 발달
    1. 메인 메모리 외에도 하드디스크 생산
    2. 메인 메모리 커짐(진공관에서 발전)
    3. 프로세스 속도 빨라짐
- 메모리에서 유저 프로그램 동작 시
    1. CPU는 빠르게 동작하나 I/O가 느림
    2. I/O실행 중 CPU idle (idle : 어떠한 프로그램에서도 사용되지 않는 상태)
- 메모리에 여러개의 프로그램을 올려 작업을 실행 (job1, job2, job3 ...)
    - CPU 유휴 시 다른 프로그램의 연산 작업 수행

      => Multiprogramming system
- 고려사항
    - 프로그램 동작 순서를 정하는 CPU scheduling 중요
    - 메모리 내에서 유저프로그램의 위치 중요

- 다른 영역의 프로그램 침범을 막기위한 보호 중요

## 4. Time-sharing system (시공유 시스템)

- 1960년대 말 Unix가 대표적
- 모니터, 키보드의 개발 => interactive system의 구현
- 컴퓨터의 가격이 비쌈 => 컴퓨터 한대에 여러대의 단말기(terminal) 연결
  - 단말기 : 모니터, 키보드만 존재




- CPU가 아주 빠르게 프로그램을 스위칭하며 동작 (Time-sharing)
- 새로운 기술의 발전
  - 유저간 통신 가능해짐
  - 동기화 : 동시에 실행 되므로 프로그램 실행 순서를 정하는 것
  - 가상메모리 : 유저가 많아지면 메인 메모리부족 => 하드디스크의 일부를 메인 메모리처럼 사용하는 기술 등장

# OS기술 천이

## 1. 컴퓨터 규모별 분류

- 1970, 1980 년대

  Supercomputer => Mainframe(단말기 수백대) => Mini(단말기 수십대) => Micro

- 현재

  Supercomputer => Server => Workstation => PC => Handheld => Embedded

## 2. 고성능 컴퓨터의 O/S기술이 handheld/embeded까지 발전

- Batch processing
- Multiprogramming
- Time-sharing (📌이번 수업 때 중점적으로 배울 것)

## 3. 고등 운영 체제의 등장

- 추후 배울 예정

# 3_고등운영체제

## 운영체제

> 강의 링크

### 3. 고등 운영체제, 인터럽트 기반 운영체제

#### A. 고등 운영체제

> 기본 운영체제

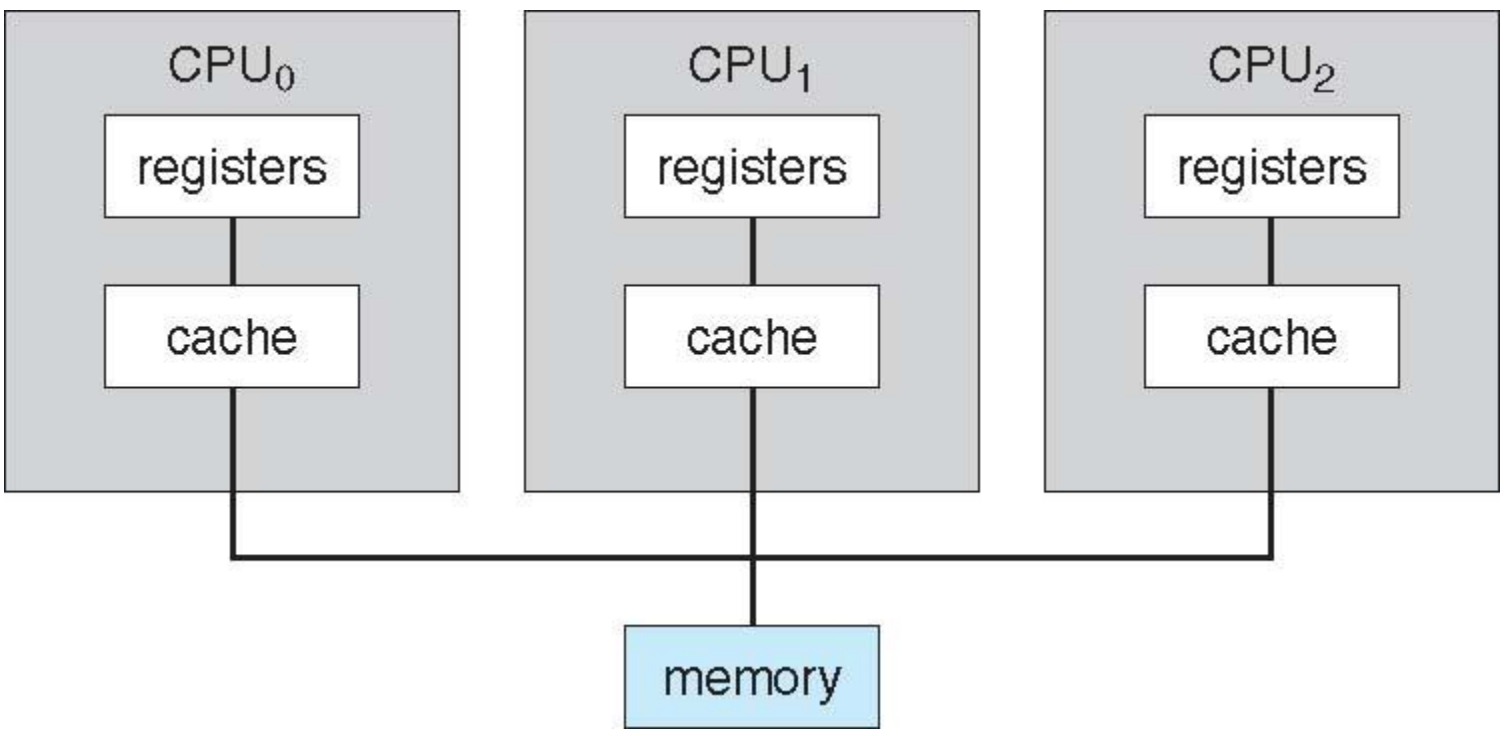
**1. 싱글 프로세서 시스템**

- 하나의 메모리에 하나의 CPU가 결합된 구조. (메모리는 Bus로 연결)

> 고등 운영체제 - 개요만 다룸

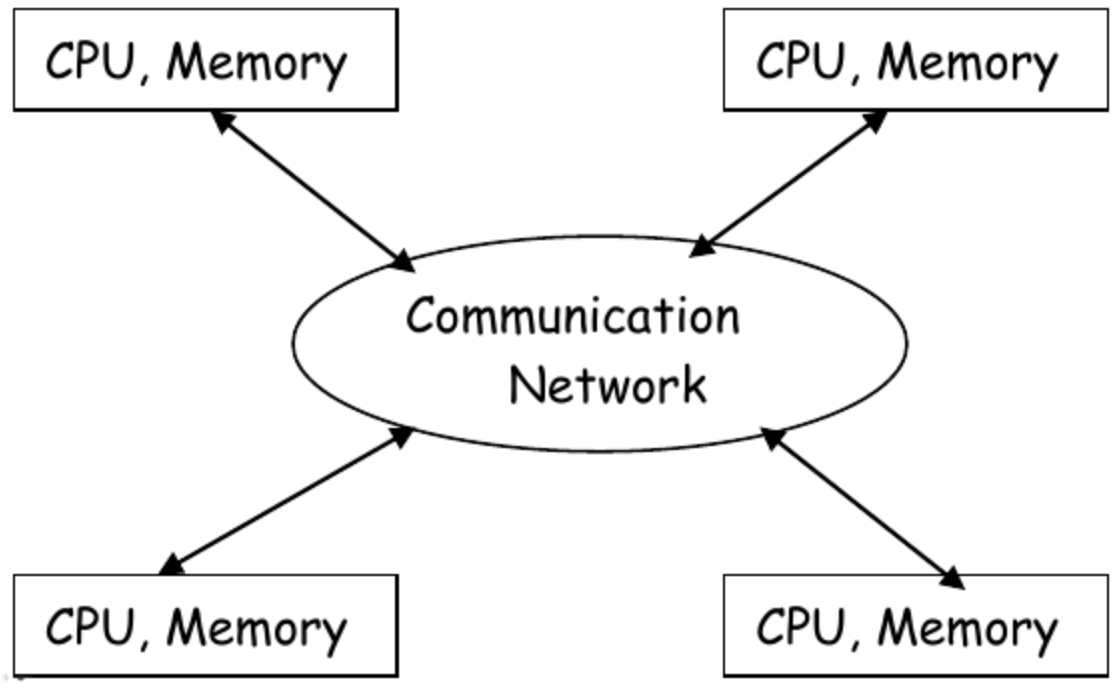
**2. 다중 프로세서 시스템(Multiprocessor system)**

- 개요
  - `병렬시스템` : 하나의 메모리에 여러개의 CPU가 병렬로 결합된 구조
  - 메인메모리에 결합되어 있기 때문에 강한 결합
  - 멀티코어(quad core 등)랑은 다름. 멀티코어는 1개의 CPU
- 장점
  - Performance : 한번에 많은 계산을 통해 성능 향상
  - Cost : 하나의 좋은 CPU보다 여러개의 싼 CPU를 사용하는게 더 저렴
  - 신뢰성 : 한개의 CPU가 고장나도 다른 CPU가 작업을 대체 할 수 있음




**3. 분산 시스템(Distributed system)**

- 개요
  - `다중 컴퓨터 시스템` : 컴퓨터 여러대를 연결해 사용하는 개념
  - LAN으로 여러 CPU-Memory 쌍(1:1)을 연결
  - 다중 프로세서 시스템과 구현목적은 같음. 비교적 느슨한 결합(메모리공유 안하다 보니까)
- 예시
  - 일기예보에서 서울, 경기, 대구 등 각 지역에 대한 분석을 각각의 컴퓨터에서 진행
  - 분석 과정에서 일부 중복되는 process는 공유하지만 다중 프로세서 시스템에 비해 느림




- **다중 프로세서 시스템 vs 분산 시스템**

```
다중 프로세서 시스템 : 1대의 컴퓨터(슈퍼컴퓨터), 동시에 여러가지 일을 처리
분산 시스템 : 여러대의 컴퓨터를 네트워크로 연결, 하나의 일을 동시에 여럿이 처리
```

#### 4. 실시간 시스템(Real-time system)

- 개요 : `Real-time OS(RTOS)` 계산이 반드시 어떤 시간내에 끝나야하는 경우
- 예시
- 더하기, 곱하기 연산은 빨리하면 좋다. 그러나 꼭 deadline이 지켜져야하는것은 아니다.
  - 내비게이션 : 교차로 진입전 방향을 결정해야됨. → 실시간 시스템 필수
- 공장자동화, 군사목적 등

**참고**

```
'리눅스는 분산시스템이고, 윈도우는 다중프로세서 시스템이다'는 틀린 문장이다. 각 OS는 어디 속해있는 개념이 아니다.
리눅스는 분산시스템과 다중프로세서 시스템의 목적으로 각각 사용될 수 있다.
```

## B. 인터럽트(가로채기) 기반 시스템

> 현대 운영체제는 인터럽트 기반 시스템

#### 운영체제(OS)

- 운영체제는 상시 메모리에 상주
- 평소에는 사건(event)을 기다리며 대기 상태에 있음
- 어떠한 명령을 지시하는 OS코드(ISR : interrupt service routine)가 내재되어있음. `마우스가 ~하면 어떤 코드를 동작시켜라`
- 운영체제의 위치는 보조기억장치(hard disk 등) 내부임

#### 인터럽트(interrupt)

- 사건을 발생시키는 행위
- 종류
  - 하드웨어 인터럽트 : 마우스 움직임, 클릭, 키보드 타이핑 등을 통해 발생된 전기신호
  - 소프트웨어 인터럽트 : 한글파일 실행, 어셈블리어(add, sub 등)
  - 내부 인터럽트 : 어떤 수를 0으로 나누지 못하므로 이때 에러 발생시키는것, 잘못실행된 프로그램을 강제로 종료하는것
- 유저의 행동에 따른 예제

  @ 한글파일 열기

  1. `처음에 컴퓨터를 킨다`

- 메인메모리는 비워져있음. CPU가 하드디스크를 뒤져 OS를 메인메로리로 가져옴 - 이 과정을 `부팅` 이라고 함.

2. `부팅 완료되어 바탕화면 도착`
   - OS는 대기상태
3. `마우스움직이면`
   - 전기신호 발생해서 CPU에 보냄(interupt)
     - > CPU는 하던일 중지하고 OS로 점프
     - > 마우스 움직이는대로 커서를 움직인다(by ISR)
4. `따블클릭`
   - 마찬가지로 interupt 발생, x,y좌표 찾아서 루틴(ISR)대로 함.
5. `한글 파일(hwp) 열기`
   - OS가 하드디스크를 뒤져 hwp 프로그램을 메인 메모리로 올림(소프트웨어 인터럽트)
     - > 한글파일에 저장된 데이터도 같이 불러옴
   - 왜 한글파일을 여는 코드는 한글파일이 아니라 ISR에 저장되어있을까 ?모든 한글파일(x.hwp)마다 이러한 코드를 저장해놓는건 메모리 낭비
6. `바탕화면에서 한글파일로 화면 전환`
   - OS는 다시 event가 생길때 까지 대기상태