※ Interrupt

Interrupt = service request to OS ← OS가 처리하는 일 중 첫번째!

service request

INT n0

INT handling

IDT (Interrupt Descriptor Table) ← ISR을 호출함!

ISR (ISR1, ISR2)

↳ Interrupt service Routine

※ OS = Σ ISR + init_code

※ IRQ (=Interrupt Request)
: 특정 작업을 하는데 있어서, 우선권이 필요한 장치들을 선언해 놓은 것 (즉 주변장치)
하드웨어 인터럽트와 관련
즉, 주변장치가 'IRQ선'을 통해 CPU에게 다가가서, "야, 특정 함수 실행시켜줘"하는 것임!

<ISR2의 함수를 호출하는 역할이 'ISR1'의 주 역할이다.>

'entry_32.S'에 위치함.

ISR2의 함수 → 실제로 인터럽트를 처리하는 함수들이 존재하는 부분

· service request:

{ application req : System call [INT : 128]

Software Interrupt.

exception (프로그램 실행 도중 발생하는 심각한 에러)

(ex) X = X/0 => [INT : 0] [INT → 0~19]

hardware req (입·출력 장치, 패킷의 도착, 컴퓨터 시간이 흐를때 마다) ← '0'으로 나눌 때

[INT → 32~47]

Hardware Interrupt

↳ 이를 바탕으로, 항상 INT이 발생하는 이유는 이것 때문이다

※ INT handling

※ INT을 디텍트하는건 cpu가 한다.

① INT x 발생!

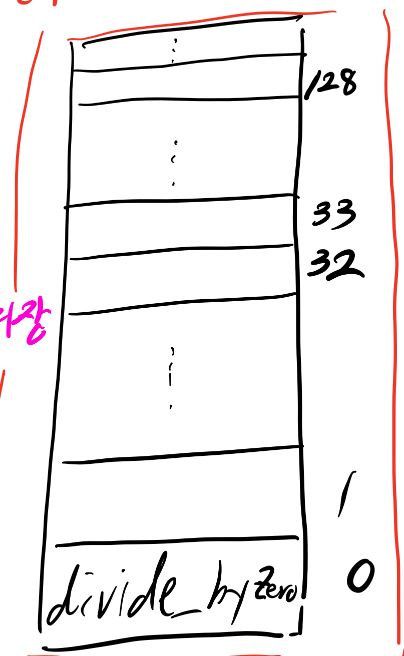
↓

주요 레지스터 저장

`push cs'
`push eip'
`push eflag'

② save cs, eip, eflag ← 현재 상태에 대한 레지스터.

③ jump to IDT[x]



↓

얘 위치는 IDT에 적혀있다.

나머지 레지스터 저장

④ ISR1 : ① save other reg
② call ISR2.
iret

`entry_32.c' 선언되어 있음.

이게 ISR1 역할 중 가장 주된 역할.

↓

⑤ ISR2 : handle INT x

⟨실제 INT를 처리하는 부분⟩

ISR2들은 전부 풀어져서 선언되어 있다.

`ISR1'이 `ISR2'의 위치를 안다.

⟨IDT(Interrupt Description Table)⟩

※ context : 현재 프로그램이 가지고 있는 모든 레지스터.

⟨※ IDT → ISR1 → ISR2⟩

ex) 키보드 press '2'

↓

INT 33발생!

↓

cpu : cs, eip, eflag 저장

① Jump IDT[33]

IDT에 'interrupt number'가 포함되어 있다.

해당 인터럽트에 대한 설명관.

↳ 'ISR1'을 호출함.

↓

② interrupt[1]:

save other reg

Jump ③ ISR2

해당 인터럽트를 처리하는 함수명이 'interrupt'라는 배열에 저장되어 있고, 이 함수를 실행하여 인터럽트를 처리한다.

↓

drivers/input/keyboard/atkbd.c

atkbd_interrupt() {

⋮

}