

## Task\_Struct :: Process의 Data Structure

- 프로세스의 속성 정보를 표현하는 가장 중요한 자료구조
- 프로세스의 이름과 PID와 같은 프로세스 정보 저장
- 프로세스의 관계를 알 수 있는 데이터 저장
- 프로세스가 생성되면 커널이 태스크 디스크립터를 프로세스에 부여

Char comm[TASK\_COMM\_LEN] → **프로세스 이름**

Pid\_t pid;  
Pid\_t tgid; → **스레드 Group ID**

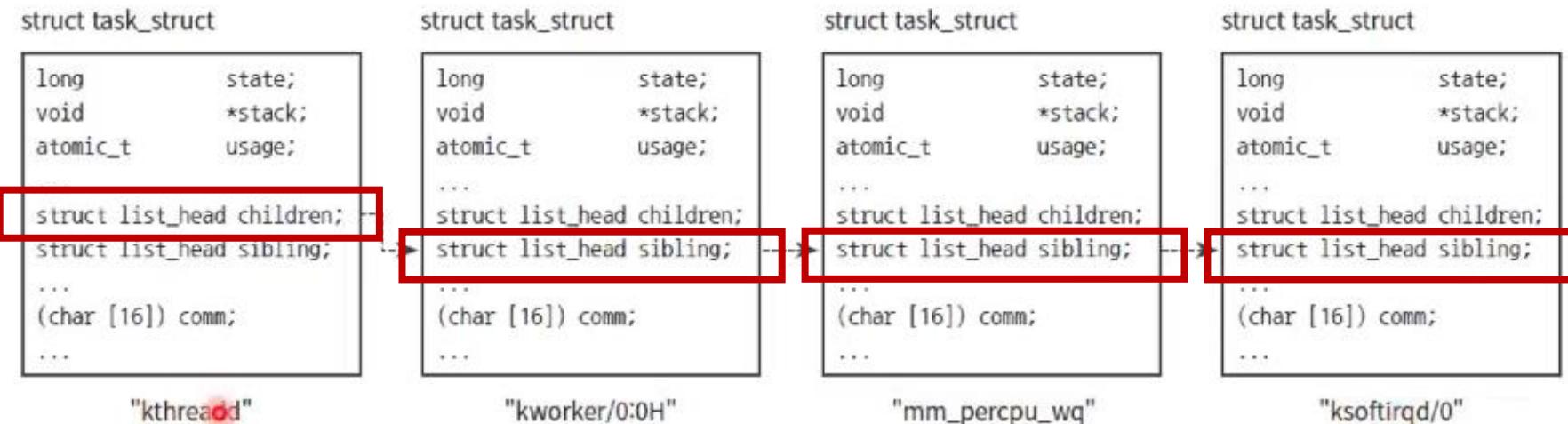
Volatile long state; → **프로세스 상태**

Struct task\_struct \*real\_parent; → **부모 태스크 디스크립터 주소**  
Struct task\_struct \*parent; → **태스크 디스크립터**

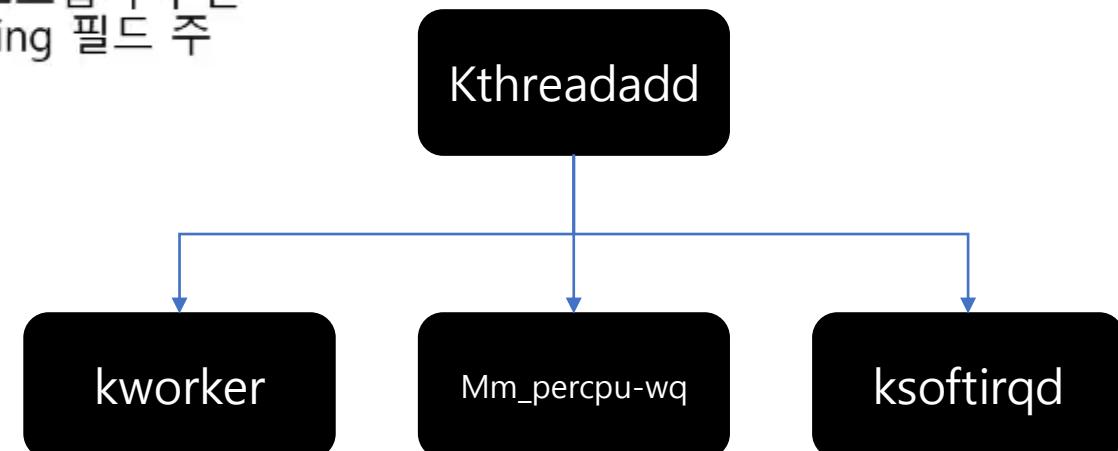
Struct list\_head children; → **children 연결 리스트에 자식 프로세스 등록**  
Struct list\_head sibling; → **형제 관계의 Process들이 등록된 연결 리스트**

# Task\_Struct :: Process의 Data Structure

"kthreadd" 프로세스의 자식 프로세스인 "kworker/0:0H" 입장에서  
"mm\_percpu\_wq"와 "ksoftirqd/0" 프로세스는 자신의 sibling 연결 리스트로 이어  
져 있음



"kthreadd" 프로세스 태스크 디스크립터의 `children` 필드는 연결 리스트입니다. 연결 리스트 헤드에 등록된 자식 프로세스의 task\_struct 구조체의 `sibling` 필드 주소를 저장



# Task\_Struct :: Process의 Data Structure

모든 Process는 Init Process의 전역 변수 필드인 tasks 연결리스트에 등록

(struct task\_struct) init\_task

|            |                               |            |
|------------|-------------------------------|------------|
| 0xA1A171B8 | (long int) state              | 0x0        |
| 0xA1A171BC | (void *) stack                | 0xA1A00000 |
| ...        | ...                           | ...        |
| 0xA1A174C8 | (struct list_head) tasks.next | 0xA1618310 |
| ...        | ...                           | ...        |

[1]

[1]: 0xA1618310 주소는 바로 연결 리스트에 등록된 다음 프로세스 태스크 디스크립터의 next 필드 주소를 의미

(struct task\_struct) 0xA1618000

|            |                               |            |
|------------|-------------------------------|------------|
| 0xA1618000 | (long int) state              | 0x1        |
| 0xA1618004 | (void *) stack                | 0xB1604000 |
| ...        | ...                           | ...        |
| 0xA1618310 | (struct list_head) tasks.next | 0xB1618A70 |
| ...        | ...                           | ...        |

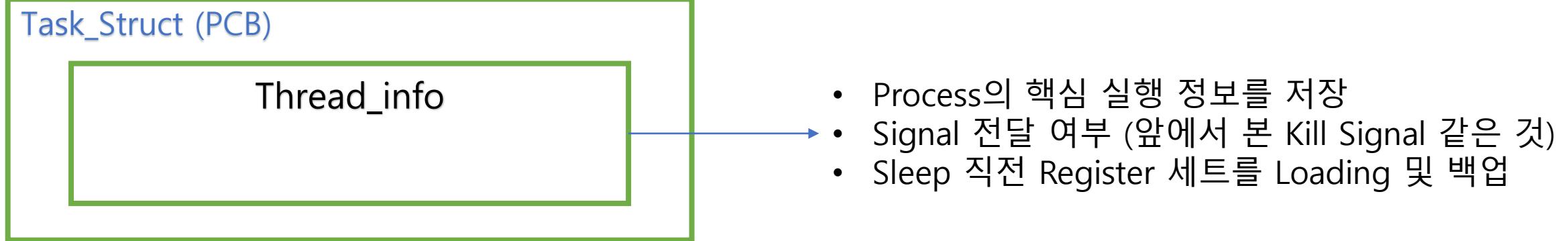
[2]

[2]: next 필드는 0xA16191B0 주소. 0xA16191B0 주소는 연결 리스트에 등록된 다음 프로세스 태스크 디스크립터의 next 필드 주소를 의미

(struct task\_struct) 0xB1618760

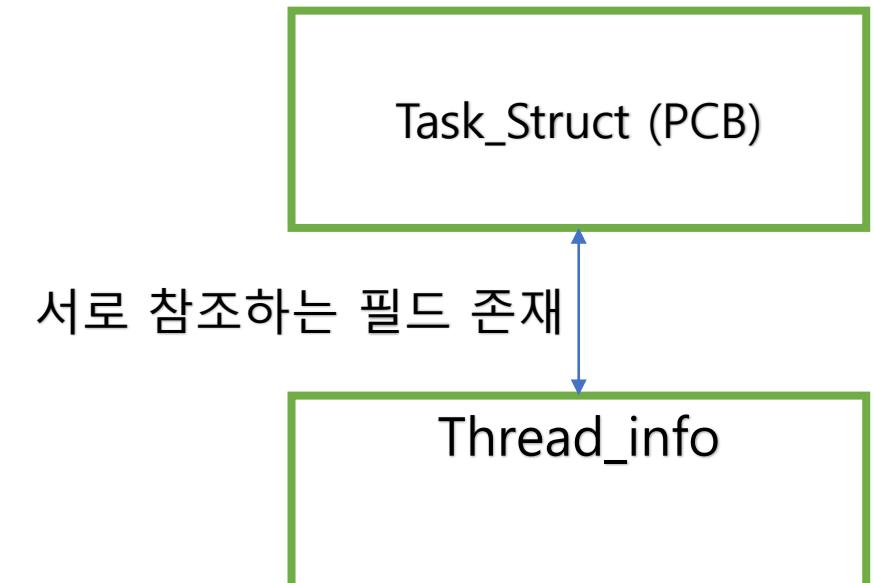
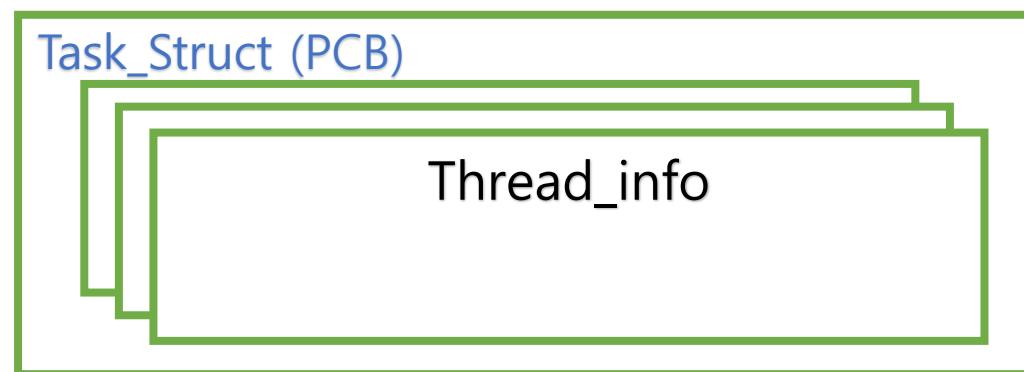
|            |                               |            |
|------------|-------------------------------|------------|
| 0xB1618760 | (long int) state              | 0x1        |
| 0xB1618764 | (void *) stack                | 0xB1604000 |
| ...        | ...                           | ...        |
| 0xB1618A70 | (struct list_head) tasks.next | 0xA16191B0 |
| ...        | ...                           | ...        |

# Thread\_info Data Structure



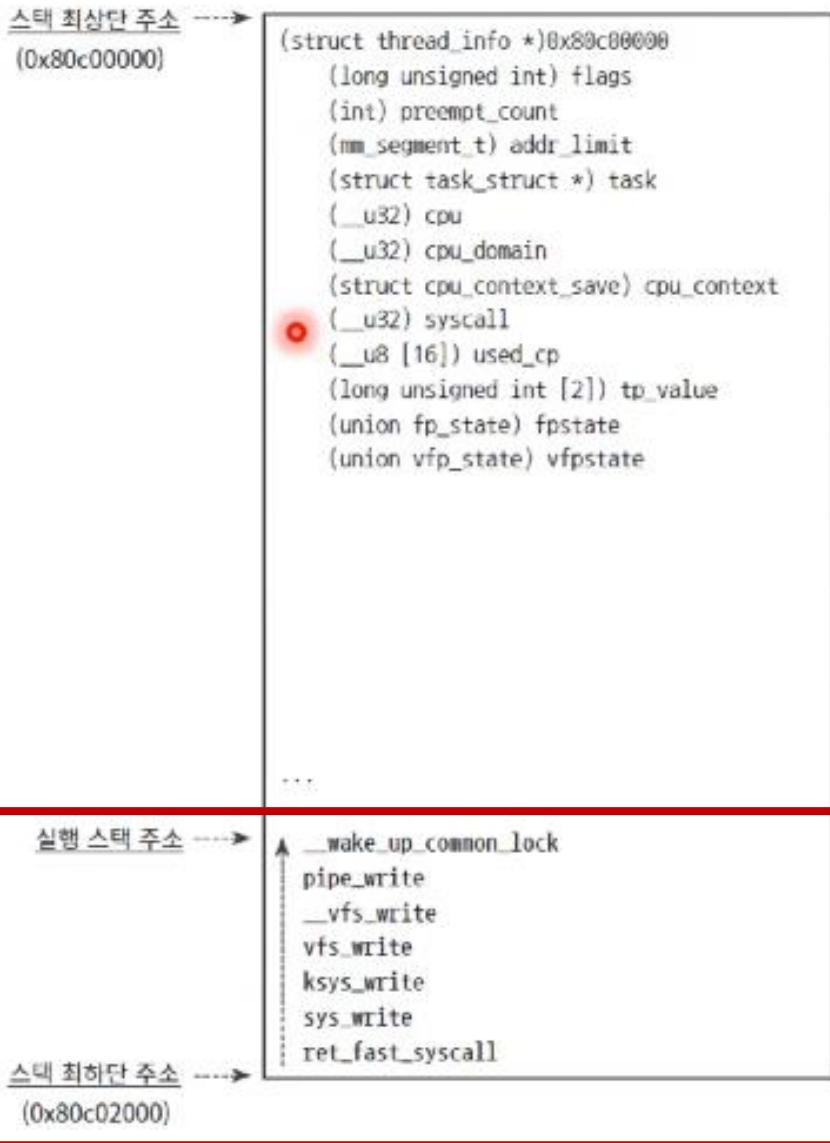
Process마다 자신의 스택이 있다. → 한 개의 Thread Info가 존재한다.

❖ 그러면, 멀티 쓰레드 환경에서는 Process마다 Thread info가 여러 개 존재하나??



# Thread\_info Data Structure

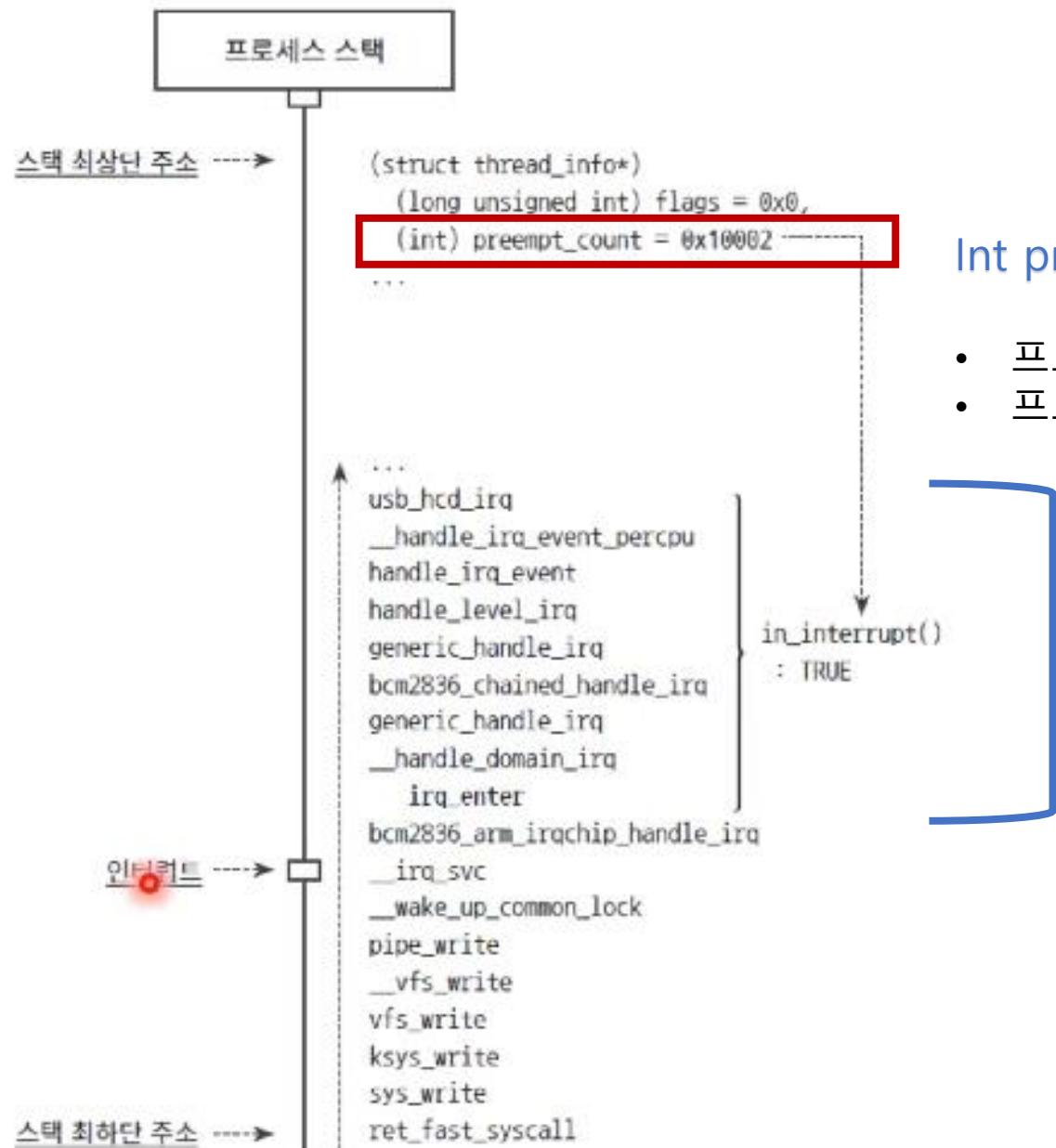
## Thread\_info



☞ 왜 Thread Info, 어떻게 생각하면 TCB에 Stack공간이 같이 있는거지?  
User Process는 User 공간에 Stack영역이 생성된다.  
Thread Info는 당연히 Kernel 공간에 존재할 것이다.  
**이것이 kernel Stack 인가? → 맞다.**

User Coder가 아닌 Kernel모드의 함수가 수행될 때, 이 공간을 사용한다.

# Thread\_info Data Structure



Int preempt\_count;

- 프로세스의 context (interrupt context, Soft IRQ context) 실행 정보
- 프로세스가 선점 스케줄링될 조건 저장

interrupt context 가 이렇게 구성된다.

- 같은 Kernel Stack을 활용하지만, Preempt\_count를 통해 Context 구분

## Current 매크로

- 현재 구동 중인 process의 task\_descriptor 주소를 알려준다.
- 따라서, task\_descriptor의 field 접근 가능

```
01 #define get_current() (current_thread_info() ->task)
02 #define current get_current()
```

