**2016025469 컴퓨터공학과 서건식**

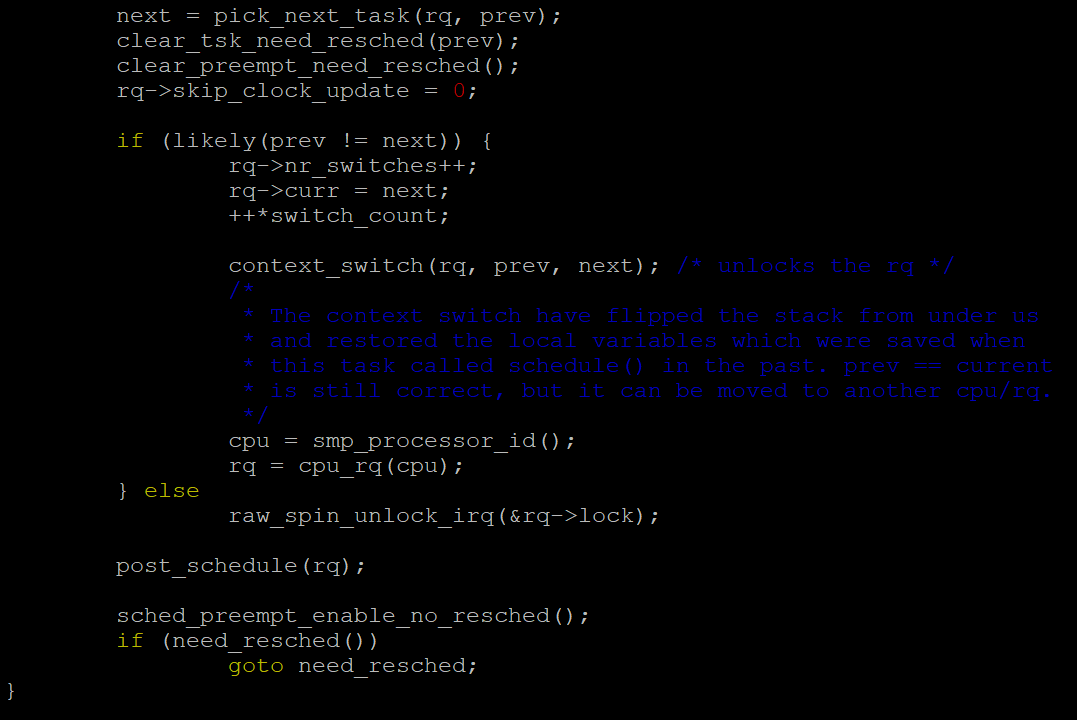
**운영 체제 HW#8**

**제출 일자 : 2020/05/21**

1. **과제 A**
2. **실습 1**

**CFS 스케줄러 기준**

1. **\_\_schedule 함수 내 Pick\_next\_task ~ \_\_schedule 함수 끝**

****

**이 부분이다.**

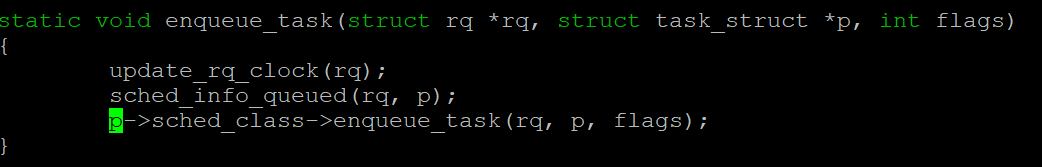
**Pick\_next\_task 함수를 통해서 현재 실행중인 태스크의 rq를 참조해 다음 태스크를 선택한다. (next에 저장) 조건문을 봤을 때 선택된 태스크가 실행되던 태스크가 아닌경우에 rq의 nr\_switches의 값을 올려주고 curr의 포인터를 다음 태스크로 바꿔준다. (현재의 태스크로 바꿔줌) 이후 context\_switch함수를 통해 context\_switching을 한다.**

**Prev와 next가 같으면 raw\_spin\_unlock\_irq함수를 호출한다.**

**이후 Post\_schedule 함수 호출->sched\_preempt\_enable\_no\_resched()**

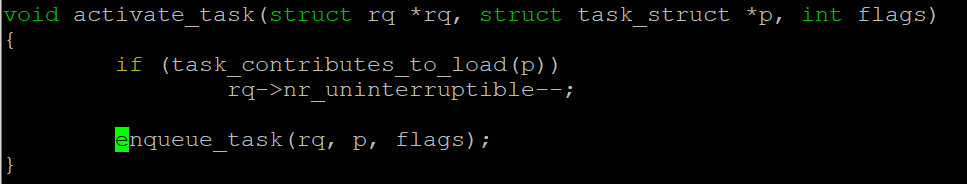
**Need\_resched()가 참이면 need\_resched로 간다. 이 함수는 스케쥴링이 다시 필요한지 검사하는 함수이다.**

1. **Do\_fork ~ p->sched\_class->enqueue\_task**

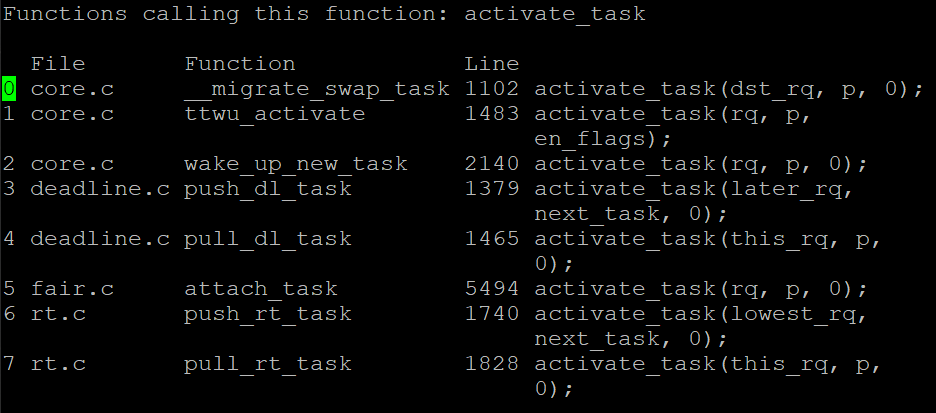
****

**역순으로 흐름을 설명하자면 이렇다.**

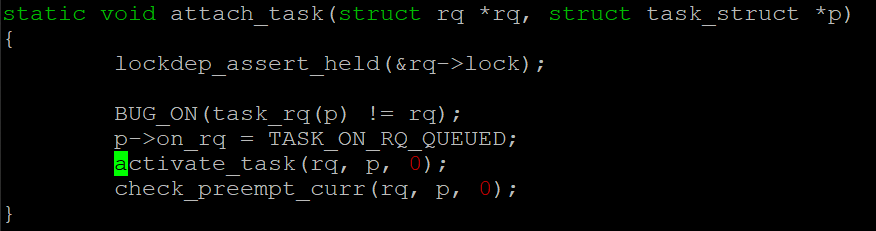
**Enqueue\_task 함수는 p->sched\_class->enqueue\_task를 호출한다. 이를 호출하는 함수를 찾아보면**

****

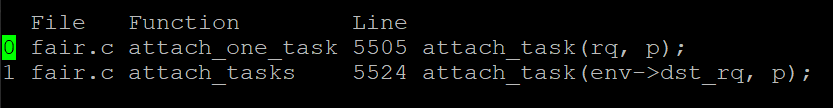
**Activate\_task가 이 함수를 호출한다. Activate\_task를 호출하는 함수를 따라가 보면**

****

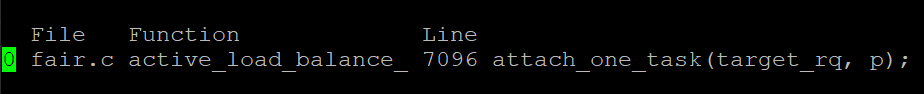
**이와 같이 많이 있는데 CFS를 보고 있으므로 fair.c의 attach\_task를 보면**

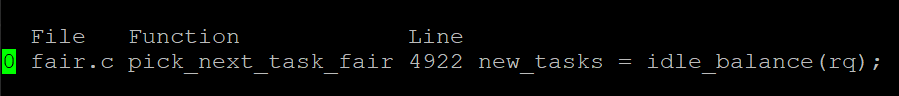
****

**이와 같다. Attach\_task를 호출하는 함수는**

****

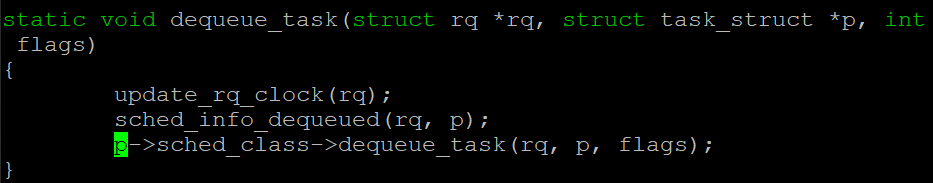
**이고, 하나를 attach하냐와 둘 이상을 attach 하냐에 따라 달라지는 함수이다.**

****

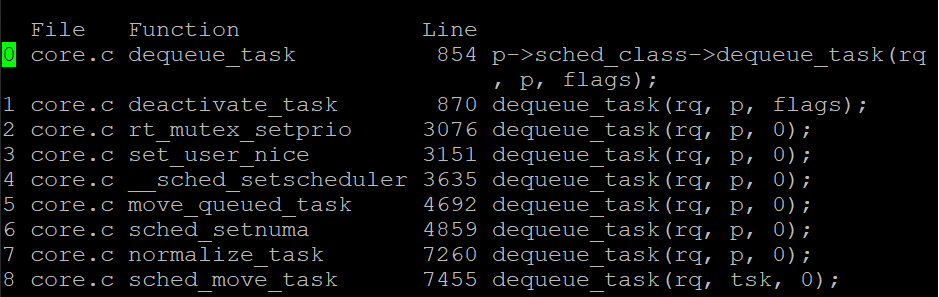
****

**Attach\_tasks -> load\_balance->idle\_balance->pick\_next\_task\_fair까지 흐름이 이어진다.**

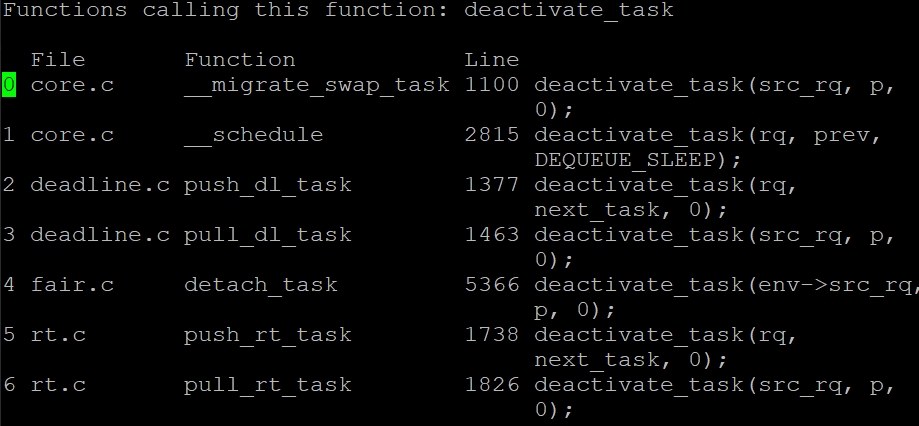
1. **\_\_schedule ~ p->sched\_class->dequeue\_task**

****

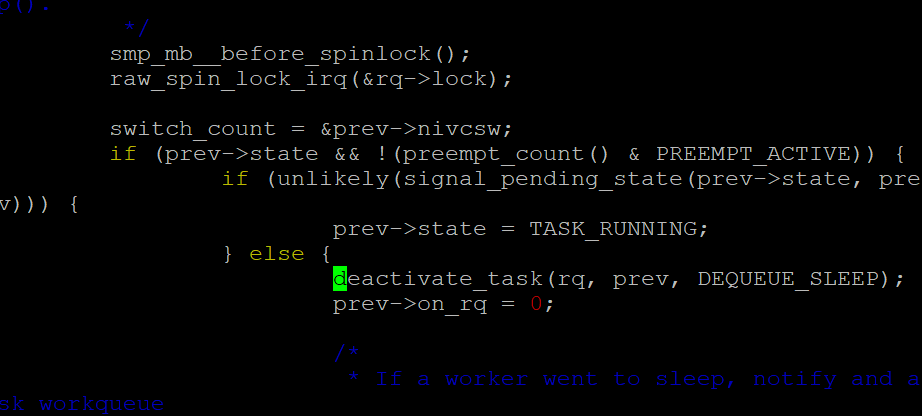
**이번에도 역순으로 들어가보면,**

****

**Dequeue\_task를 호출하는 함수는 다음과 같다. Deactivate\_task를 호출하는 함수를 살펴보면**

****

**다음과 같다. 우리는 \_\_schedule함수 내에서 찾아야 하므로 다음 함수를 들어가 코드를 살펴보면**

****

**다음과 같다.**

1. **리눅스 스케줄러 전체 동작과 mysched 설명**

**전체동작:**

**전체적인 스케줄링 과정은 다음과 같다. Exit, wait상태의 태스크들이 RUNNING state로 전환하게 되면 rq 에 enqueue\_task()함수를 사용해서 삽입해 준다. Pick\_next\_task()함수로 rq의 태스크 중 다음으로 수행할 태스크를 산출하여 rq->curr로 해주고, 이를 실행한다. Rq->curr 상태의 태스크가 RUNNING 상태로 돌아올 땐 put\_prev\_task(), EXIT,WAIT등 다른 상태로 돌아갈 땐 dequeue\_task() 함수를 호출한다.**

**Mysched 설명:**

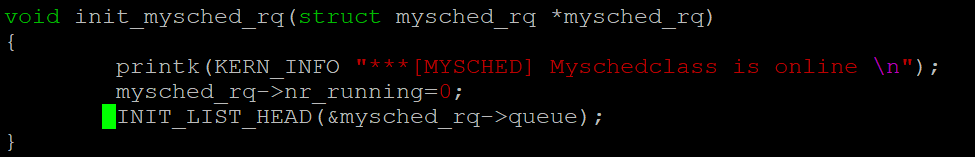
**Mysched 서브 런큐는 double Linked list로 구현했다.**

**그룹 스케쥴링의 위치는 cfs와 idle 사이에 존재한다. 스케쥴링의 정책은 FIFO이다.**

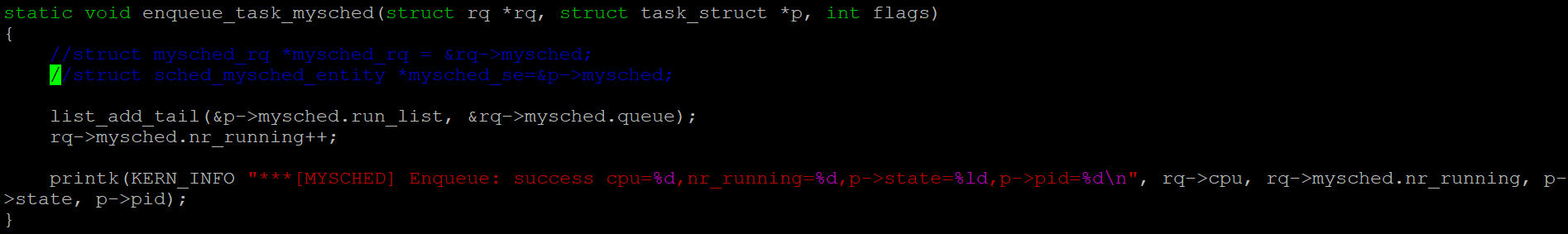
**Sched class의 동작을 설명하면, enqueue\_task\_mysched는 서브런큐의 마지막에 태스크의 sched\_mysched\_entity를 삽입 후 nr\_running 값을 1 증가 시킨다. Dequeue\_task\_mysched는 반대로 삭제 후 nr\_running 값을 1 감소시킨다.**

**Pick\_next\_task\_mysched 는 mysched rq가 비어있지 않으면 pick\_next\_task 과정을 계속 진행한다. Put\_prev\_task 호출 후 sub rq에서 sched\_mysched\_entity 원소 하나를 pick해 그 부모 task\_struct를 찾아 반환함.(container of 활용)**

1. **제공된 mysched.c 설명**

****

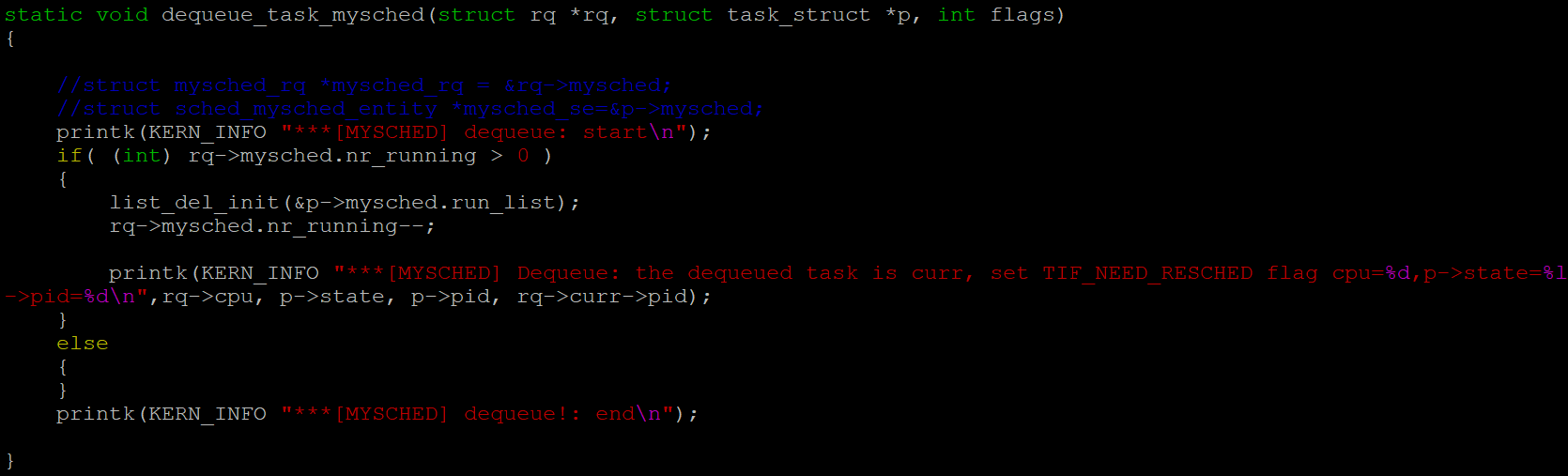
**Rq를 초기화 하는 함수이다.**

****

**Enqueue\_task\_mysched**

**실행가능 상태의 태스크를 스케줄러의 서브런큐에 삽입하는 함수. Rq와 p, flag를 인자로 받는다.**

**List\_add\_tail로 linked list의 tail에 process를 추가한다. 또한 인자로 받은 rq의 nr\_running값을 하나 올려준다.**

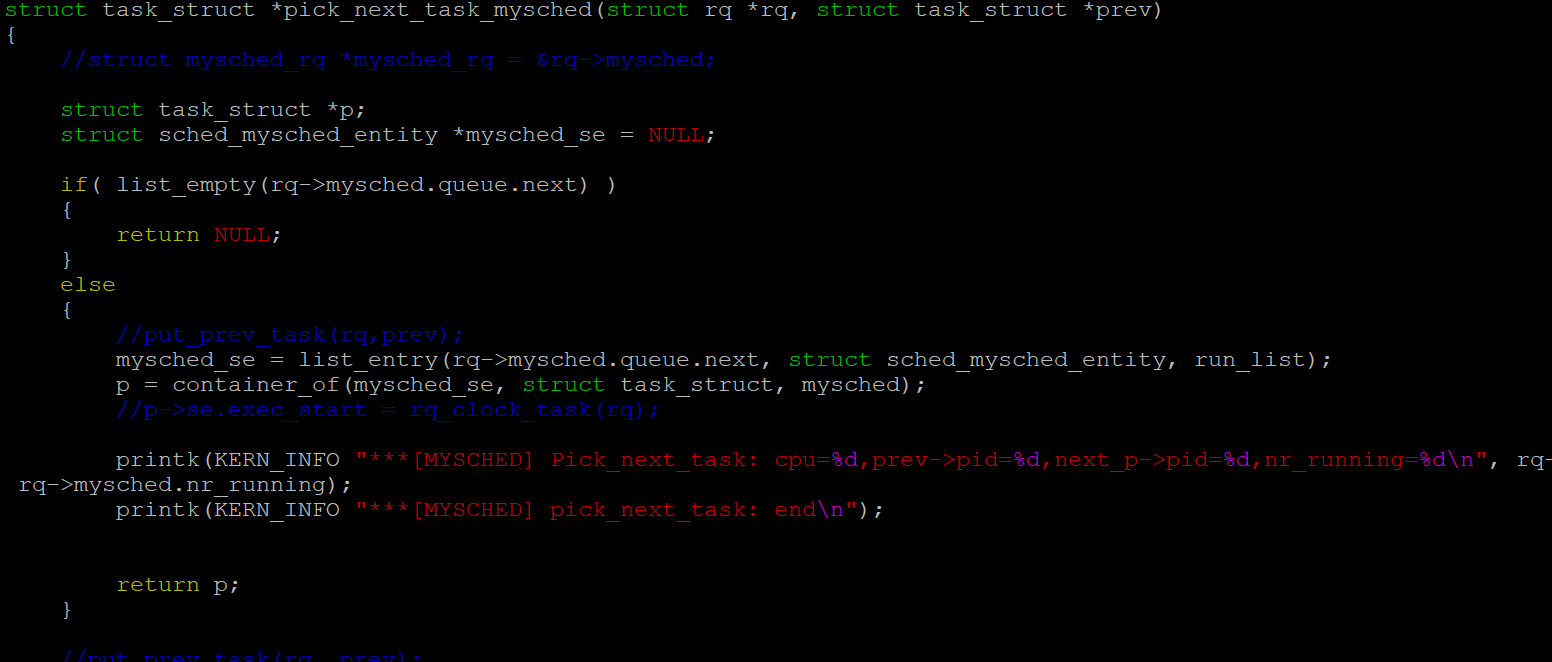
****

**Dequeue\_task\_mysched**

**실행중인 태스크가 exit, 실행 불가능 상태로 전환되면 rq에서 태스크를 삭제하는 함수이다.**

**첫 if문은 running 중인 태스크가 존재할 때 (0보다 크면 하나 이상 존재)**

**List\_del\_init함수로 태스크를 삭제한 후 nr\_running값을 줄여준다(태스크 하나 삭제)**

****

**Pick\_next\_task\_mysched**

**다음 수행할 태스크를 선택하는 함수이다.**

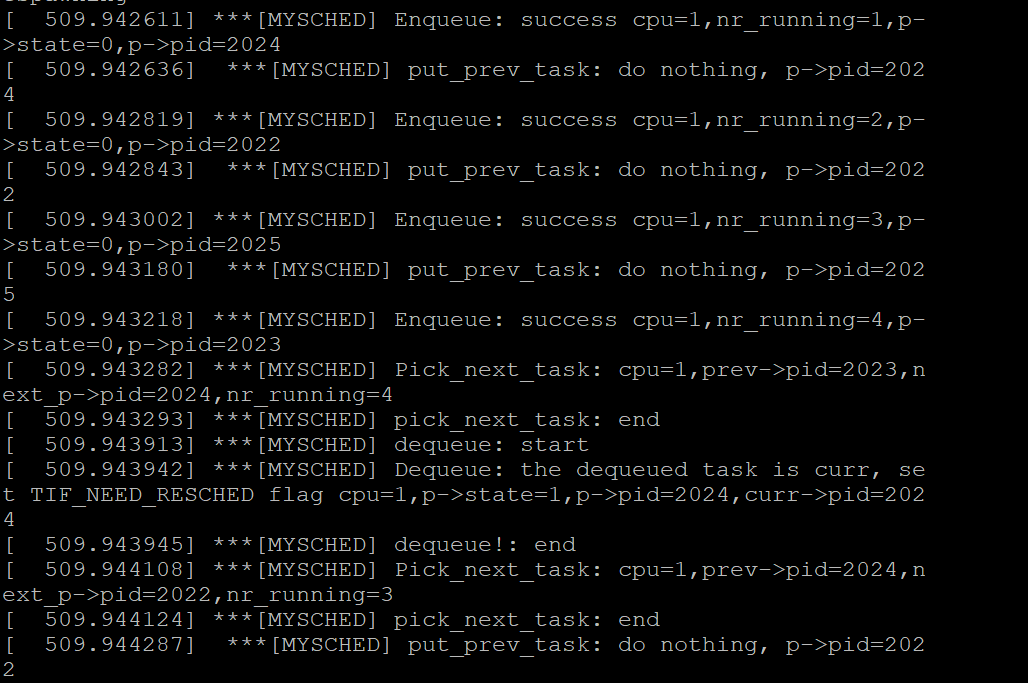
**Rq와 전에 수행중이던 태스크를 인자로 받는다.**

**조건문으로 rq의 다음 태스크가 있는지 유무를 파악 후, 없다면 NULL을 반환한다.**

**있다면 다음 태스크의 값을 entity, run\_list,mysched.queue.next를 인자로 받아 받은 후에 p에 저장한다. (container\_of 사용)**

**그 후 p를 리턴한다.(next task)**

1. **최종 결과**

****