2016025469 컴퓨터공학과 서건식

운영 체제 HW#8

제출 일자 : 2020/05/21

A. 과제 A

1. 실습 1

CFS 스케줄러 기준

1) __schedule 함수 내 Pick_next_task ~ __schedule 함수 끝

```
        next = pick_next_task(rq, prev);
        clear_tsk_need_resched(prev);
        clear_preempt_need_resched();
        rq->skip_clock_update = 0;

        if (likely(prev != next)) {
                rq->nr_switches++;
                rq->curr = next;
                ++*switch_count;

                context_switch(rq, prev, next); /* unlocks the rq */
                /*
                 * The context switch have flipped the stack from under us
                 * and restored the local variables which were saved when
                 * this task called schedule() in the past. prev == current
                 * is still correct, but it can be moved to another cpu/rq.
                 */
                cpu = smp_processor_id();
                rq = cpu_rq(cpu);
        } else
                raw_spin_unlock_irq(&rq->lock);

        post_schedule(rq);

        sched_preempt_enable_no_resched();
        if (need_resched())
                goto need_resched;
}
```

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.

2) Do_fork ~ p->sched_class->enqueue_task

```
static void enqueue_task(struct rq *rq, struct task_struct *p, int flags)
{
        update_rq_clock(rq);
        sched_info_queued(rq, p);
        p->sched_class->enqueue_task(rq, p, flags);
}
```

역순으로 흐름을 설명하자면 이렇다.

Enqueue_task 함수는 p->sched_class->enqueue_task를 호출한다. 이를 호출하는 함수를 찾아보면

```
void activate_task(struct rq *rq, struct task_struct *p, int flags)
{
        if (task_contributes_to_load(p))
                rq->nr_uninterruptible--;

        enqueue_task(rq, p, flags);
}
```

Activate_task가 이 함수를 호출한다. Activate_task를 호출하는 함수를 따라가 보면

```
Functions calling this function: activate_task

  File        Function             Line
0 core.c      __migrate_swap_task  1102 activate_task(dst_rq, p, 0);
1 core.c      ttwu_activate        1483 activate_task(rq, p,
                                        en_flags);
2 core.c      wake_up_new_task     2140 activate_task(rq, p, 0);
3 deadline.c push_dl_task          1379 activate_task(later_rq,
                                        next_task, 0);
4 deadline.c pull_dl_task          1465 activate_task(this_rq, p,
                                        0);
5 fair.c      attach_task          5494 activate_task(rq, p, 0);
6 rt.c        push_rt_task         1740 activate_task(lowest_rq,
                                        next_task, 0);
7 rt.c        pull_rt_task         1828 activate_task(this_rq, p,
                                        0);
```

이와 같이 많이 있는데 CFS를 보고 있으므로 fair.c의 attach_task를 보면

```
static void attach_task(struct rq *rq, struct task_struct *p)
{
        lockdep_assert_held(&rq->lock);

        BUG_ON(task_rq(p) != rq);
        p->on_rq = TASK_ON_RQ_QUEUED;
        activate_task(rq, p, 0);
        check_preempt_curr(rq, p, 0);
}
```

이와 같다. Attach_task를 호출하는 함수는

```
  File    Function         Line
0 fair.c attach_one_task 5505 attach_task(rq, p);
1 fair.c attach_tasks    5524 attach_task(env->dst_rq, p);
```

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

```
  File    Function             Line
0 fair.c active_load_balance_ 7096 attach_one_task(target_rq, p);
```

```
  File    Function             Line
0 fair.c pick_next_task_fair 4922 new_tasks = idle_balance(rq);
```

Attach_tasks -> load_balance->idle_balance->pick_next_task_fair까지 흐름이 이어진다.

3) __schedule ~ p->sched_class->dequeue_task

```
static void dequeue_task(struct rq *rq, struct task_struct *p, int
 flags)
{
        update_rq_clock(rq);
        sched_info_dequeued(rq, p);
        p->sched_class->dequeue_task(rq, p, flags);
}
```

이번에도 역순으로 들어가보면,

```
  File    Function              Line
0 core.c dequeue_task           854 p->sched_class->dequeue_task(rq
                                    , p, flags);
1 core.c deactivate_task        870 dequeue_task(rq, p, flags);
2 core.c rt_mutex_setprio      3076 dequeue_task(rq, p, 0);
3 core.c set_user_nice         3151 dequeue_task(rq, p, 0);
4 core.c __sched_setscheduler  3635 dequeue_task(rq, p, 0);
5 core.c move_queued_task      4692 dequeue_task(rq, p, 0);
6 core.c sched_setnuma         4859 dequeue_task(rq, p, 0);
7 core.c normalize_task        7260 dequeue_task(rq, p, 0);
8 core.c sched_move_task       7455 dequeue_task(rq, tsk, 0);
```

Dequeue_task를 호출하는 함수는 다음과 같다. Deactivate_task를 호출하는 함수를 살펴보면

```
Functions calling this function: deactivate_task

  File        Function            Line
0 core.c      __migrate_swap_task 1100 deactivate_task(src_rq, p,
                                       0);
1 core.c      __schedule          2815 deactivate_task(rq, prev,
                                       DEQUEUE_SLEEP);
2 deadline.c push_dl_task         1377 deactivate_task(rq,
                                       next_task, 0);
3 deadline.c pull_dl_task         1463 deactivate_task(src_rq, p,
                                       0);
4 fair.c      detach_task         5366 deactivate_task(env->src_rq,
                                       p, 0);
5 rt.c        push_rt_task        1738 deactivate_task(rq,
                                       next_task, 0);
6 rt.c        pull_rt_task        1826 deactivate_task(src_rq, p,
                                       0);
```

다음과 같다. 우리는 __schedule함수 내에서 찾아야 하므로 다음 함수를 들어가 코드를 살펴보면

```
p().
         */
        smp_mb__before_spinlock();
        raw_spin_lock_irq(&rq->lock);

        switch_count = &prev->nivcsw;
        if (prev->state && !(preempt_count() & PREEMPT_ACTIVE)) {
                if (unlikely(signal_pending_state(prev->state, pre
v))) {
                        prev->state = TASK_RUNNING;
                } else {
                        deactivate_task(rq, prev, DEQUEUE_SLEEP);
                        prev->on_rq = 0;

                        /*
                         * If a worker went to sleep, notify and a
sk workqueue
```

다음과 같다.

2. 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)

3. 제공된 mysched.c 설명

```
void init_mysched_rq(struct mysched_rq *mysched_rq)
{
        printk(KERN_INFO "***[MYSCHED] Myschedclass is online \n");
        mysched_rq->nr_running=0;
        INIT_LIST_HEAD(&mysched_rq->queue);
}
```

Rq를 초기화 하는 함수이다.

```
static void enqueue_task_mysched(struct rq *rq, struct task_struct *p, int flags)
{
    //struct mysched_rq *mysched_rq = &rq->mysched;
    //struct sched_mysched_entity *mysched_se=&p->mysched;

    list_add_tail(&p->mysched.run_list, &rq->mysched.queue);
    rq->mysched.nr_running++;

    printk(KERN_INFO "***[MYSCHED] Enqueue: success cpu=%d,nr_running=%d,p->state=%ld,p->pid=%d\n", rq->cpu, rq->mysched.nr_running, p-
>state, p->pid);
}
```

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.

```
static void dequeue_task_mysched(struct rq *rq, struct task_struct *p, int flags)
{

    //struct mysched_rq *mysched_rq = &rq->mysched;
    //struct sched_mysched_entity *mysched_se=&p->mysched;
    printk(KERN_INFO "***[MYSCHED] dequeue: start\n");
    if( (int) rq->mysched.nr_running > 0 )
    {
        list_del_init(&p->mysched.run_list);
        rq->mysched.nr_running--;

        printk(KERN_INFO "***[MYSCHED] Dequeue: the dequeued task is curr, set TIF_NEED_RESCHED flag cpu=%d,p->state=%l
->pid=%d\n",rq->cpu, p->state, p->pid, rq->curr->pid);
    }
    else
    {
    }
    printk(KERN_INFO "***[MYSCHED] dequeue!: end\n");

}
```

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)

```
struct task_struct *pick_next_task_mysched(struct rq *rq, struct task_struct *prev)
{
    //struct mysched_rq *mysched_rq = &rq->mysched;

    struct task_struct *p;
    struct sched_mysched_entity *mysched_se = NULL;

    if( list_empty(rq->mysched.queue.next) )
    {
        return NULL;
    }
    else
    {
        //put_prev_task(rq,prev);
        mysched_se = list_entry(rq->mysched.queue.next, struct sched_mysched_entity, run_list);
        p = container_of(mysched_se, struct task_struct, mysched);
        //p->se.exec_start = rq_clock_task(rq);

        printk(KERN_INFO "***[MYSCHED] Pick_next_task: cpu=%d,prev->pid=%d,next_p->pid=%d,nr_running=%d\n", rq-
 rq->mysched.nr_running);
        printk(KERN_INFO "***[MYSCHED] pick_next_task: end\n");


        return p;
    }

    //put_prev_task(rq, prev);
```

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)

4. 최종 결과

```
[  509.942611] ***[MYSCHED] Enqueue: success cpu=1,nr_running=1,p-
>state=0,p->pid=2024
[  509.942636]  ***[MYSCHED] put_prev_task: do nothing, p->pid=202
4
[  509.942819] ***[MYSCHED] Enqueue: success cpu=1,nr_running=2,p-
>state=0,p->pid=2022
[  509.942843]  ***[MYSCHED] put_prev_task: do nothing, p->pid=202
2
[  509.943002] ***[MYSCHED] Enqueue: success cpu=1,nr_running=3,p-
>state=0,p->pid=2025
[  509.943180]  ***[MYSCHED] put_prev_task: do nothing, p->pid=202
5
[  509.943218] ***[MYSCHED] Enqueue: success cpu=1,nr_running=4,p-
>state=0,p->pid=2023
[  509.943282] ***[MYSCHED] Pick_next_task: cpu=1,prev->pid=2023,n
ext_p->pid=2024,nr_running=4
[  509.943293] ***[MYSCHED] pick_next_task: end
[  509.943913] ***[MYSCHED] dequeue: start
[  509.943942] ***[MYSCHED] Dequeue: the dequeued task is curr, se
t TIF_NEED_RESCHED flag cpu=1,p->state=1,p->pid=2024,curr->pid=202
4
[  509.943945] ***[MYSCHED] dequeue!: end
[  509.944108] ***[MYSCHED] Pick_next_task: cpu=1,prev->pid=2024,n
ext_p->pid=2022,nr_running=3
[  509.944124] ***[MYSCHED] pick_next_task: end
[  509.944287]  ***[MYSCHED] put_prev_task: do nothing, p->pid=202
2
```