2018008331 컴퓨터소프트웨어학부 박민경

운영 체제 HW#9

제출 일자 : 2020/05/31

A. 과제 A

1. 실습 1

X

2. 리눅스 스케줄러 전체 동작과 myrr 설명

RT 스케줄러에서 RT task의 우선순위는 0~99로 나뉜다. 우선순위별로 linked list로 구현된 run queue가 존재하며, 0으로 갈수록 우선 순위가 높아 먼저 실행된다. update_curr_rt를 통해 할당된 시간을 초과하면 플래그를 set하여 스케줄링이 일어나도록 하는 방식이다. pick_next_task로 다음 태스크가 뽑히면 스케줄러에 의해 rq->curr로 설정된다. myrr 스케줄러는 리눅스가 제공하는 linked list를 사용하여 run queue를 구현한다. 스케줄러 우선순위는 rt->mysched->myrr->cfs->idle로 설정하며, Time Slice를 정확한 값으로 사용하기 어렵기 때문에 update_curr_rt 함수를 세 번 실행하면 스케줄링 되는 방식이다. task_tick 함수가 하드웨어 틱에 의해 호출되는데, 일정하게 호출되기가 어렵기 때문에 task_tick에 의해 호출되는 update_curr_myrr도 불규칙 할 수 있다. fork된 프로세스들이 작업을 수행하다가 할당된 시간(값)을 초과하면 스케줄링이 일어나는데, 이 때 실행하던 프로세스는 서브런큐의 마지막 node가 되고 그 프로세스의 next가 head의 next가 되어 다음 태스크가 된다.

3. myrr.c 설명

myrr.c는 연속적인 pid를 가지는 myrr 프로세스들이 newclass 코드에 의해 1초 간격으로 fork되고, fork된 각각의 프로세스는 1씩 200000000번 더하는 작업을 10번 반복한다. 그리고 모든 작업이 끝난 뒤 프로세스를 종료하고, 프로세스의 update_num이 3보다 크면 reschedule 함수를 호출하여 스케줄링을 실행한다.

자료구조로는 구조체 sched_myrr_entity, myrr_rq, task_struct, rq를 정의한다. sched_myrr_entity 구조체는 멤버로 run_list와 update_num을 가지며 각각은 entity의 list 노드, task_tick을 위한 update_num을 가진다. myrr_rq는 서브런큐의 head인 list 노드 head와 enqueue된 프로세스의 개수 nr_running을 멤버로 가진다. task_struct는 sched_myrr_entity 구조체형 멤버를 가지며, rq는 myrr_rq 구조체형 멤버를 가진다.

```
void init_myrr_rq (struct myrr_rq *myrr_rq)
{
        printk(KERN_INFO "***[MYRR] Mysched class is online \n");
        myrr_rq->nr_running=0;
        INIT_LIST_HEAD(&myrr_rq->queue);

}
```

init_myrr_rq 함수는 런큐 myrr_rq를 초기화한다. 초기에는 myrr_rq가 비어 있으므로 nr_running 값을 0으로 초기화하고, INIT_LIST_HEAD 함수를 호출하여 list head인 myrr_rq->queue를 초기화한다.

```
static void update_curr_myrr(struct rq *rq){

        struct task_struct *p = rq->curr;
        struct sched_myrr_entity *myrr_se = &p->myrr;
        struct list_head *lh = &rq->curr->myrr.run_list;
        myrr_se->update_num++;
        if(myrr_se->update_num>3)
        {
                list_del_init(lh);
                list_add_tail(lh, &rq->myrr.queue);
                myrr_se->update_num = 0;
                printk(KERN_INFO"***[MYRRUPDATECUREND] update_curr_end\tpid=%d u
pdate_num=%d\n", p->pid, myrr_se->update_num);
                resched_curr(rq);
        }
        else
        {
                printk(KERN_INFO"***[MYRR] update_curr_myrr\t pid=%d update_num=
%d\n", p->pid,myrr_se->update_num);
        }
}
```

update_curr_myrr은 task_tick에 의해 호출되는 함수로, 현재 태스크의 정보를 업데이트하고, 일정 조건 하에서 리스케줄을 요청한다.

호출될 때마다 현재 태스크의 update_num을 1 증가시키고 만약

update_num이 3을 초과할 시, 서브런큐의 entity를 재배열한다. 현재 실행되고 있던 프로세스를 prev라 하면, prev의 next가 head의 next가 되고, prev는 서브런큐의 맨 마지막 노드가 되며 prev의 update_num은 다시 0으로 초기화한다. 그리고 나서 resched_curr 함수를 호출하여 리스케줄링을 요청한다.

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

        list_add_tail(&p->myrr.run_list, &rq->myrr.queue);

        rq->myrr.nr_running++;
        printk(KERN_INFO"***[MYRR] enqueue: success cpu=%d, nr_running=%d, pid=%
d\n",cpu_of(rq), rq->myrr.nr_running,p->pid);
}
```

enqueue_task_myrr 함수는 myrr 서브런큐 맨 마지막에 태스크를 추가하기 위해 list_add_tail 함수를 호출한다. 그리고 서브런큐의 nr_running 값을 1 증가시켜 현재 enqueue되어 있는 프로세스의 개수를 업데이트한다.

```
static void dequeue_task_myrr(struct rq *rq, struct task_struct *p, int flags)
{
        if(rq->myrr.nr_running>0)
        {

                list_del_init(&p->myrr.run_list);
                rq->myrr.nr_running--;
                printk(KERN_INFO"\t***[MYRR] dequeue: success cpu=%d, nr_running
=%d, pid=%d\n",cpu_of(rq), rq->myrr.nr_running,p->pid);
        }
        else{
        }

}
```

dequeue_task_myrr 함수는 myrr 서브런큐에 태스크가 존재할 경우 현재 실행중인 태스크를 dequeue하기 위해 list_del_init 함수를 호출한다. 그리고 서브런큐의 nr_running 값을 1 감소시켜 현재 enqueue되어 있는 프로세스의 개수를 업데이트한다.

```
struct task_struct *pick_next_task_myrr(struct rq *rq, struct task_struct *prev)
{

        struct sched_myrr_entity *myrr_se = NULL;
        struct task_struct *p;
        if(rq->myrr.nr_running==0 ){
                return NULL;
        }

        myrr_se = list_entry(rq->myrr.queue.next, struct sched_myrr_entity, run_
list);
        p = container_of(myrr_se, struct task_struct, myrr);

        printk(KERN_INFO "\t***[MYRR] pick_next_task: cpu=%d, prev->pid=%d,next_
p->pid=%d,nr_running=%d\n",cpu_of(rq),prev->pid,p->pid,rq->myrr.nr_running);
        return p;

}
```

pick_next_task_myrr 함수는 다음에 실행될 태스크를 선출한다. 만약 서브런큐에 어떤 태스크도 enqueue되어 있지 않다면, NULL을 return한다. 그렇지 않을 경우 서브런큐 head인 queue의 next, 즉 list의 첫번째 태스크를 다음 task로 선출하여 해당 run_list를 가지고 있는 부모 구조체를 찾아 task_struct형으로 리턴한다.

그 이외에 task_tick_myrr 함수는 update_curr_myrr 함수를 호출함을 통해 time slice를 이용한 스케줄링을 한다.

4. 최종 결과

```
[   22.529341] ***[MYRR] update_curr_myrr        pid=1877 update_num=1
[   22.529561]  ***[MYRR] dequeue: success cpu=1, nr_running=3, pid=1877
[   22.529564]  ***[MYRR] put_prev_task: do_nothing, p->pid=1877
[   22.529566] ***[MYRR] set_curr_task_myrr
[   22.529570] ***[MYRR] enqueue: success cpu=1, nr_running=4, pid=1877
[   22.529591]  ***[MYRR] dequeue: success cpu=1, nr_running=3, pid=1877
[   22.529594]  ***[MYRR] pick_next_task: cpu=1, prev->pid=1877,next_p->pid=1880
,nr_running=3
[   22.529610]  ***[MYRR] dequeue: success cpu=1, nr_running=2, pid=1880
[   22.529612]  ***[MYRR] put_prev_task: do_nothing, p->pid=1880
[   22.529614] ***[MYRR] set_curr_task_myrr
[   22.529636] ***[MYRR] enqueue: success cpu=1, nr_running=3, pid=1880
[   22.529642]  ***[MYRR] dequeue: success cpu=1, nr_running=2, pid=1880
[   22.529644]  ***[MYRR] pick_next_task: cpu=1, prev->pid=1880,next_p->pid=1878
,nr_running=2
[   22.529659]  ***[MYRR] dequeue: success cpu=1, nr_running=1, pid=1878
[   22.529661]  ***[MYRR] put_prev_task: do_nothing, p->pid=1878
[   22.529663] ***[MYRR] set_curr_task_myrr
[   22.529667] ***[MYRR] enqueue: success cpu=1, nr_running=2, pid=1878
[   22.529673]  ***[MYRR] dequeue: success cpu=1, nr_running=1, pid=1878
[   22.529675]  ***[MYRR] pick_next_task: cpu=1, prev->pid=1878,next_p->pid=1879
,nr_running=1
```

<출력 1>

```
,nr_running=4
[   22.470712] ***[MYRR] update_curr_myrr        pid=1880 update_num=1
[   22.472031] ***[MYRR] update_curr_myrr        pid=1880 update_num=2
[   22.473037] ***[MYRR] update_curr_myrr        pid=1880 update_num=3
[   22.474037] ***[MYRR] update_curr_myrr        pid=1880 update_num=4
[   22.474123] ***[MYRRUPDATECUREND] update_curr_end    pid=1880 update_num=4
[   22.474317]  ***[MYRR] pick_next_task: cpu=1, prev->pid=1880,next_p->pid=1878
,nr_running=4
[   22.474676] ***[MYRR] update_curr_myrr        pid=1878 update_num=1
[   22.475986] ***[MYRR] update_curr_myrr        pid=1878 update_num=2
[   22.477115] ***[MYRR] update_curr_myrr        pid=1878 update_num=3
[   22.478055] ***[MYRR] update_curr_myrr        pid=1878 update_num=4
[   22.478136] ***[MYRRUPDATECUREND] update_curr_end    pid=1878 update_num=4
[   22.478342]  ***[MYRR] pick_next_task: cpu=1, prev->pid=1878,next_p->pid=1879
,nr_running=4
[   22.478359] ***[MYRR] update_curr_myrr        pid=1879 update_num=1
[   22.479284] ***[MYRR] update_curr_myrr        pid=1879 update_num=2
[   22.480922] ***[MYRR] update_curr_myrr        pid=1879 update_num=3
[   22.482507] ***[MYRR] update_curr_myrr        pid=1879 update_num=4
[   22.482648] ***[MYRRUPDATECUREND] update_curr_end    pid=1879 update_num=4
[   22.483081] ***[MYRR] update_curr_myrr        pid=1879 update_num=1
[   22.483178]  ***[MYRR] pick_next_task: cpu=1, prev->pid=1879,next_p->pid=1877
,nr_running=4
[   22.483862] ***[MYRR] update_curr_myrr        pid=1877 update_num=2
[   22.484864] ***[MYRR] update_curr_myrr        pid=1877 update_num=3
[   22.485848] ***[MYRR] update_curr_myrr        pid=1877 update_num=4
[   22.485912] ***[MYRRUPDATECUREND] update_curr_end    pid=1877 update_num=4
[   22.486059]  ***[MYRR] pick_next_task: cpu=1, prev->pid=1877,next_p->pid=1880
,nr_running=4
```

<출력 2>

update_curr_task가 4번 호출되었을 경우, time slice를 소진했다고 판단하여 resched_curr를 호출해서 flag를 set하고, pick_next 함수에 의해 선출된 다음 태스크를 실행한다. 여기서 즉시 스케줄 함수가 일어나 pick_next 함수가 호출되지는 않기 때문에 update_curr가 한번 추가적으로 실행되는 경우가 발생한다.

https://drive.google.com/file/d/1DF_eY9wgEkvRl0kWWIO-RRkGdQ68QkgY/view?usp=sharing