1. 리눅스 스케줄러 전체 동작과 myrr 설명

➔ Linux 스케줄러의 전체 우선 순위는 stop, dl, rt, mysched, myrr, fair, idle 순으로 정했기 때문에 myrr보다 높은 우선순위 스케줄러의 런큐가 비어 있는 경우 myrr의 런큐 속의 task에 대한 스케줄링이 시작됩니다. 우선 task가 enqueue되면 자신의 속성에 맞는 스케줄러의 서브 런큐로 들어가게 됩니다. 각각의 서브 런큐의 정책에 맞게 실행되다가 자신의 task를 모두 수행하거나 sleep되는 경우 dequeue되며 외의 경우에는 다시 자신이 있던 서브 런큐로 돌아가 자신의 차례를 기다리게 됩니다.

➔ Myrr의 정책: myrr은 round robin 방식으로 주어진 time slice를 모두 소진하면 CPU 점유를 그만두고 서브 런큐로 돌아가게 됩니다. 이때 큐의 맨 뒤에 삽입되게 되고 동시에 rescheduling을 요청합니다. 우리의 과제에서는 시간을 사용할 경우 정확한 결과를 보기 힘들기 때문에 update_num이라는 변수를 이용하여 3번보다 많이 수행되면 CPU 점유를 그만두는 정책을 사용하였습니다. 따라서 task가 자신의 일을 모두 수행하거나 Sleep되는 경우 dequeue되고 아닌 경우 myrr의 정책에 맞게 서브 런큐의 맨 뒤로 돌아가고 rescheduling을 요청하게 됩니다.

2. myrr.c 파일에 대한 설명

```
#define MYRR_TIME_SLICE 3
```

➔ time slice를 3으로 설정했습니다

```
static void update_curr_myrr(struct rq *rq){
        /*
        if running task use every time slice which it has,
        1. reschedule (change turn in sub run queue
        2. initialize update num
        3. use resched_curr(rq) for rescheduling
        */
        struct myrr_rq *myrr_rq = &rq->myrr;
        struct sched_myrr_entity *myrr_se = list_entry(myrr_rq->queue.next, struct sched_myrr_entity, run_list);
        struct task_struct *p = container_of(myrr_se, struct task_struct, myrr);

        myrr_se->update_num++;

        printk(KERN_INFO"***[MYRR] update_curr_myrr             pid=%d update_num=%d\n", p->pid, myrr_se->update_num);

        if (myrr_se->update_num > MYRR_TIME_SLICE) {
                myrr_se->update_num = 0;

                // TODO: reschedule - change turn in sub rq
                // Delete first and Insert old first
                list_del(&p->myrr.run_list);
                list_add_tail(&p->myrr.run_list, &rq->myrr.queue);

                resched_curr(rq);

                printk(KERN_INFO"***[MYRR] update_curr_myrr: Change curr\n");
        }

}
```

➔ update_num을 확인하여 3보다 큰 경우 queue의 안의 1번째 task를 맨 뒤로 보내고 rescheduling을 요청하며 3보다 작거나 같은 경우 update_num을 1만큼 증가시키는 함수입니다.

➔ myrr_rq: 현재 myrr의 rq입니다

➔ myrr_se: 현재 myrr의 entity입니다

➔ p: 현재 task_struct입니다

➔ myrr_se->update_num++: 현재 task의 수행 횟수를 늘려줍니다

➔ myrr_se->update_num이 time slice보다 많은 경우

1. myrr_se->update_num을 0으로 만듦
2. list_del을 사용하여 내용은 지우지 않고 list에서만 삭제
3. list_add_tail을 이용하여 삭제한 run_list를 다시 서브 런큐의 맨 뒤에 연결
4. 위의 과정을 통해 서브 런큐의 2번째 task가 1번째로 변경되고 원래의 1번째 task가 맨 뒤로 이동하게 됩니다.
5. resched_curr(rq)를 이용해 rescheduling 요청

```
static void enqueue_task_myrr(struct rq *rq, struct task_struct *p, int flags) {
        // TODO: Push to sub rq
        printk(KERN_INFO"***[MYRR] Enqueue Start\n");

        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);
        printk(KERN_INFO"***[MYRR] Enqueue End\n");
}
```

➔ task가 들어오면 myrr의 서브 런큐로 task를 삽입시켜주는 함수입니다

➔ list_add_tail을 이용하여 서브 런큐의 맨 뒤에 현재 task를 연결

➔ rq->myrr.nr_running++로 현재 실행중인 task 개수 증가시킴

```
static void dequeue_task_myrr(struct rq *rq, struct task_struct *p, int flags)
{
        printk(KERN_INFO "***[MYRR] Dequeue: start\n");

        if(rq->myrr.nr_running > 0) {
                /*
                OSDC Lab. Add code
                1. operate dequeue in sub rq
                */

                // Dequeue in sub rq
                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{}

        printk(KERN_INFO "***[MYRR] Dequeue: end\n");

}
```

➔ task의 동작이 완료되거나 sleep된 경우 서브 런큐에서 task를 삭제하는 함수입니다.

➔ 현재 수행중인 task가 있는 경우

1. List_del_init을 사용하여 list에서 현재 task를 내용 초기화하여 삭제
2. rq->myrr.nr_running--로 현재 실행중인 task 개수 감소시킴

```
struct task_struct *pick_next_task_myrr(struct rq *rq, struct task_struct *prev)
{
        struct myrr_rq *myrr_rq = &rq->myrr;
        struct sched_myrr_entity *myrr_se = NULL;
        struct task_struct *p;

        if(rq->myrr.nr_running == 0 ){
                return NULL;
        }

        else {
                //TODO: pick next task from sub rq
                update_curr_myrr(rq);

                myrr_se = list_entry(myrr_rq->queue.next, struct sched_myrr_entity, run_list);
                p = container_of(myrr_se, struct task_struct, myrr);
                printk(KERN_INFO "\t\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);
                printk(KERN_INFO"***[MYRR] Pick Next Task Myrr End\n");
                return p;
        }
}
```

➔ CPU가 정책에 맞게 task를 수행한 뒤 다음 task를 뽑기 위해 수행되는 함수입니다

➔ myrr_rq: 현재 myrr의 rq입니다

➔ 수행중인 task가 0개가 아닌 경우

1. Update_curr_myrr(rq)를 실행

2. myrr_se는 위의 update_curr_myrr(rq) 이전에 수행된(= resched_curr(rq)를 호출한) update_curr_myrr(rq)에 의해 2번째에서 1번째로 바뀐 task의 entity를 받아옵니다

3. P는 myse의 task_struct이고 이를 return합니다

3. 최종 결과

```
[ 2983.766443] ***[MYRR] update_curr_myrr               pid=1951 update_num=1
[ 2983.766473]          ***[MYRR] Pick_next_task: cpu=1, prev->pid=1952, next_p->pid=1951, nr_running=4
[ 2983.766501] ***[MYRR] Pick Next Task Myrr End
[ 2983.766892] ***[MYRR] update_curr_myrr               pid=1951 update_num=2
[ 2983.767863] ***[MYRR] update_curr_myrr               pid=1951 update_num=3
[ 2983.769236] ***[MYRR] update_curr_myrr               pid=1951 update_num=4
[ 2983.769279] ***[MYRR] update_curr_myrr: Change curr
[ 2983.769403] ***[MYRR] update_curr_myrr               pid=1954 update_num=1
[ 2983.769431]          ***[MYRR] Pick_next_task: cpu=1, prev->pid=1951, next_p->pid=1954, nr_running=4
[ 2983.769456] ***[MYRR] Pick Next Task Myrr End
[ 2983.769884] ***[MYRR] update_curr_myrr               pid=1954 update_num=2
[ 2983.770856] ***[MYRR] update_curr_myrr               pid=1954 update_num=3
[ 2983.771853] ***[MYRR] update_curr_myrr               pid=1954 update_num=4
[ 2983.771855] ***[MYRR] update_curr_myrr: Change curr
[ 2983.771867] ***[MYRR] update_curr_myrr               pid=1953 update_num=1
[ 2983.771869]          ***[MYRR] Pick_next_task: cpu=1, prev->pid=1954, next_p->pid=1953, nr_running=4
[ 2983.771872] ***[MYRR] Pick Next Task Myrr End
[ 2983.773421] ***[MYRR] update_curr_myrr               pid=1953 update_num=2
[ 2983.774625] ***[MYRR] update_curr_myrr               pid=1953 update_num=3
[ 2983.774932] ***[MYRR] update_curr_myrr               pid=1953 update_num=4
[ 2983.774935] ***[MYRR] update_curr_myrr: Change curr
[ 2983.774947] ***[MYRR] update_curr_myrr               pid=1952 update_num=1
[ 2983.774950]          ***[MYRR] Pick_next_task: cpu=1, prev->pid=1953, next_p->pid=1952, nr_running=4
[ 2983.774952] ***[MYRR] Pick Next Task Myrr End
[ 2983.775846] ***[MYRR] update_curr_myrr               pid=1952 update_num=2
[ 2983.776856] ***[MYRR] update_curr_myrr               pid=1952 update_num=3
[ 2983.777850] ***[MYRR] update_curr_myrr               pid=1952 update_num=4
[ 2983.777853] ***[MYRR] update_curr_myrr: Change curr
[ 2983.777866] ***[MYRR] update_curr_myrr               pid=1951 update_num=1
[ 2983.777869]          ***[MYRR] Pick_next_task: cpu=1, prev->pid=1952, next_p->pid=1951, nr_running=4
[ 2983.777871] ***[MYRR] Pick Next Task Myrr End
[ 2983.779429] ***[MYRR] update_curr_myrr               pid=1951 update_num=2
[ 2983.779897] ***[MYRR] update_curr_myrr               pid=1951 update_num=3
[ 2983.780846] ***[MYRR] update_curr_myrr               pid=1951 update_num=4
[ 2983.780849] ***[MYRR] update_curr_myrr: Change curr
[ 2983.780862] ***[MYRR] update_curr_myrr               pid=1954 update_num=1
[ 2983.780865]          ***[MYRR] Pick_next_task: cpu=1, prev->pid=1951, next_p->pid=1954, nr_running=4
[ 2983.780868] ***[MYRR] Pick Next Task Myrr End
[ 2983.782344] ***[MYRR] update_curr_myrr               pid=1954 update_num=2
[ 2983.782894] ***[MYRR] update_curr_myrr               pid=1954 update_num=3
```

➔ 제일 위의 메시지에서 pid=1951이 선택되어 update_curr_myrr 수행되어 update_num이 1만큼 증가하고 pick_next_task 메시지 출력

➔ Task가 3번 수행된 뒤 4번째에서 Change curr 발생

➔ 위와 동일한 logic으로 계속 수행됨

➔ 1951 -> 1954 -> 1953 ->1952 -> 1951 -> 1954 -> ....의 순서로 3번씩 실행되는 것을 보아 myrr 정책에 맞게 수행된다고 볼 수 있습니다

4. OVA 링크

➔ https://drive.google.com/open?id=1Ed09SNEr2VN1uyuWdHiLX8nTocaqdBmh