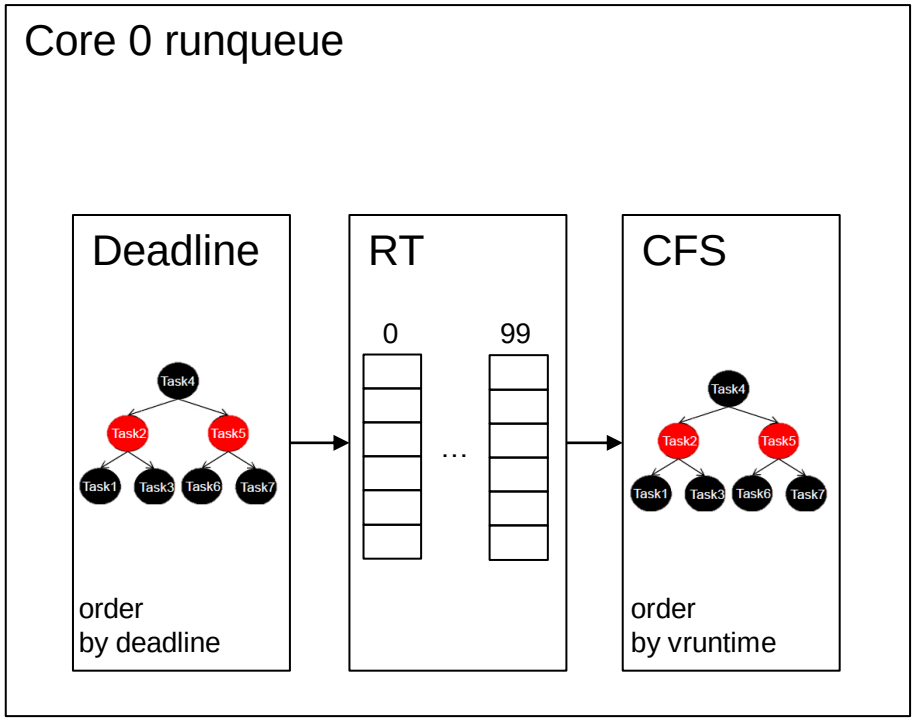
RT/ Round-Robin Scheduler

A. Task_tick() & update_curr()

Round Robin과 관련된 sched_class 함수

- **RT sched class의 RR 정책**
  - **RT Sched class엔 두 가지 정책이 존재하고, 정책에 따라 동작**
    1. Sched_RR
    2. Sched_FIFO



Round Robin과 관련된 sched_class 함수

위치: linux-3.18.27/kernel/sched/sched.h

```
1088  struct sched_class {
1089          const struct sched_class *next;
1090
1091          void (*enqueue_task) (struct rq *rq, struct task_struct *p, int flags);
1092          void (*dequeue_task) (struct rq *rq, struct task_struct *p, int flags);
1106          struct task_struct * (*pick_next_task) (struct rq *rq,
1107                                                  struct task_struct *prev);
1108          void (*put_prev_task) (struct rq *rq, struct task_struct *p);
1109
1125          void (*set_curr_task) (struct rq *rq);
1126          void (*task_tick) (struct rq *rq, struct task_struct *p, int queued);
1138          void (*update_curr) (struct rq *rq);
1143  };
```

다음 task를 고르기 전에 보관해야 될 task를 선정

<struct sched_class>

현재 time slice에 대한 시간 값을 update

ex) RT 스케줄러의 update_curr은 얼마나 시간을 할당해주는것은

현재 task가 실행되고 있고, time slice로 쪼갠 얼마시간이 언제인지 알려줌

또는 정해진 초기값에서부터 현재 시간까지의 차이를 계산

task_tick

- task_tick
  - 일정 시간마다 주기적으로 실행됨 ← 하드웨어에서 주기적으로 Interrupt를 건다
  - 현재 실행 중인 태스크의 실행 시간과 할당된 타임 슬라이스 값을 비교하는 함수
  - 할당된 시간을 모두 소진했을 경우 해당 태스크에 'TIF_NEED_RESCHED'이라는 플래그를 SET하여 스케줄링이 가능하도록 설정하며 이는 resched_curr 함수를 통해 이루어짐
  - 태스크의 실행 시간 업데이트는 update_curr를 통해 이루어지며 task_tick에서 실행 시간 업데이트를 할 때 update_curr를 보통 사용

```
void scheduler_tick(void)
{
        int cpu = smp_processor_id();
        struct rq *rq = cpu_rq(cpu);
        struct task_struct *curr = rq->curr;
                        :
                        :
        update_rq_clock(rq);
        curr->sched_class->task_tick(rq, curr, 0);
        update_cpu_load_active(rq);
```

or

```
static void task_tick_dl(struct rq *rq, struct task_struct *p, int queued)
{
        update_curr_dl(rq);
```

or

```
static void task_tick_rt(struct rq *rq, struct task_struct *p, int queued)
{
        struct sched_rt_entity *rt_se = &p->rt;
                                :
                                :
        resched_curr(rq);
```

```
static void task_tick_fair(struct rq *rq, struct task_struct *curr, int queued)
{
        struct cfs_rq *cfs_rq;
        struct sched_entity *se = &curr->se;
                        if (queued) {
                                resched_curr(rq_of(cfs_rq));
                                return;
                                :
                                :
```

<각 태스크가 속한 클래스별 함수로 동작>

update_curr

- task_tick 함수에서 curr의 실행시간을 업데이트 할 때 사용하는 함수
  - sched_entity(cfs)의 경우 exec_start, sum_exec_time을 update_curr에서 업데이트
  - deadline과 rt 스케줄러는 cfs가 아니지만 sched_entity(cfs)의 멤버변수를 사용하여 시간값 사용

처음 함수를 초기화 했을 때부터 task가 얼마동안 실행됐는지 측정

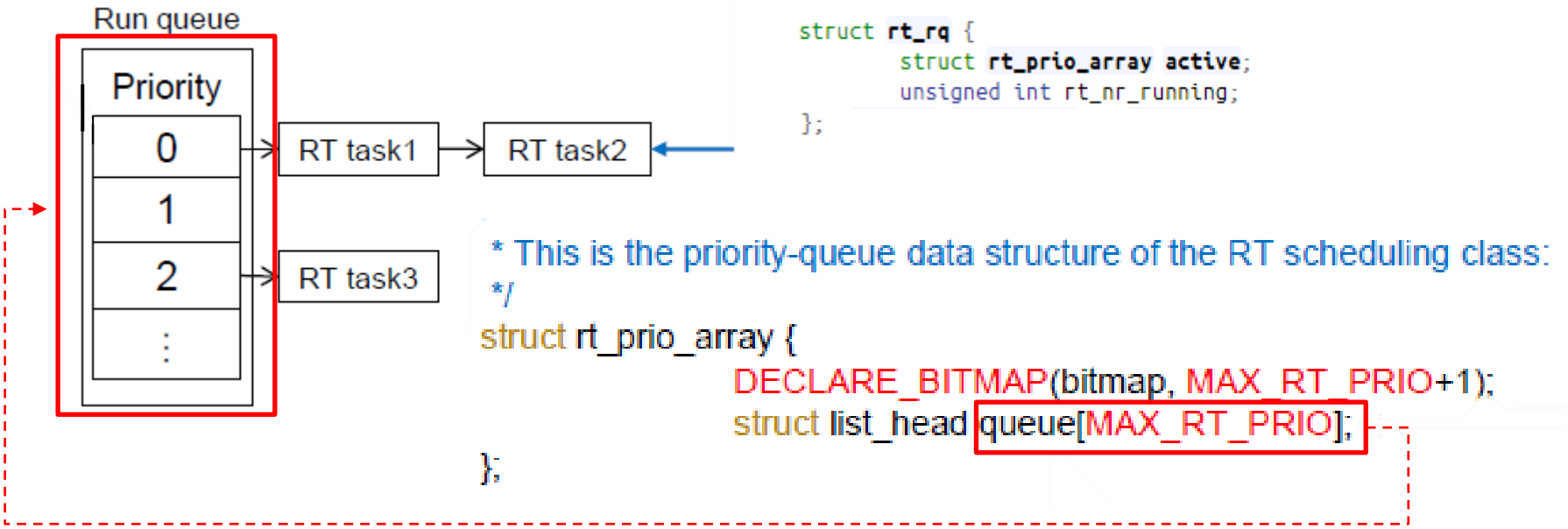
```
static void task_tick_rt(struct rq *rq, struct task_struct *p, int queued)
{
        struct sched_rt_entity *rt_se = &p->rt;

        update_curr_rt(rq);
}
```

```
static void update_curr_rt(struct rq *rq)
{
        struct task_struct *curr = rq->curr;
        struct sched_rt_entity *rt_se = &curr->rt;
        u64 delta_exec;

        if (curr->sched_class != &rt_sched_class)
                return;

        delta_exec = rq_clock_task(rq) - curr->se.exec_start;
}
```

B. RT 스케줄러의 실행시간 업데이트 동작방식

RT 스케줄러

- **RT task의 우선 순위**
  - Rt task의 우선순위는 0~99로 나뉨
  - 우선순위별로 linked list로 구현된 run queue가 존재
  - 0으로 갈수록 우선 순위가 더 높아 먼저 실행됨
- **RT 태스크용 스케줄링 정책(policy)**
  - SCHED_FIFO: 먼저 실행된 태스크가 끝날 때 까지 계속 수행
  - SCHED_RR: 같은 우선 순위의 태스크는 커널에서 설정된 기간(디폴트 100ms) 단위로 순서를 바꿈



<RT 스케줄러의 서브런큐>

RT 스케줄러의 update_curr 동작방식

- update_curr를 통해 시간을 업데이트하는 스케줄러는 deadline, rt, cfs가 있음
  - 시간과 관련된 변수는 sched_entity의 멤버변수 sum_exec_time과 exec_start를 사용
    - sum_exec_time: 태스크가 실행을 시작한 후부터 지금까지 실행한 총 시간
    - exec_start: 마지막으로 sum_exec_runtime을 업데이트했던 시각
  - sum_exec_time과 exec_start를 통해 태스크가 실행한 시간을 계산함
- update_curr_rt의 동작방식
  - 1. 마지막 업데이트 클럭 시간부터 현재 클럭 시간까지의 시간 차이 계산
  - 2. sum_exec_time(rq->rt_time)에 시간 차이만큼 더해줌
  - 3. exec_start를 rq_clock_task 값으로 업데이트
  - 4. 만약 sum_exec_time(rq->rt_time)이 할당된 시간을 초과하면 TIF_NEED_RESCHED 플래그를 set하여 스케줄링이 일어나게 함

```
static void update_curr_rt(struct rq *rq)
{
        struct task_struct *curr = rq->curr;
        struct sched_rt_entity *rt_se = &curr->rt;
        u64 delta_exec;
1 <-----delta_exec = rq_clock_task(rq) - curr->se.exec_start;
2 <-----curr->se.sum_exec_runtime += delta_exec;
3 <-----curr->se.exec_start = rq_clock_task(rq);
4 <-----if (sched_rt_runtime_exceeded(rt_rq))
                resched_curr(rq);
}
```

<update_curr_rt>

RT 스케줄러의 update_curr 동작방식

- pick_next_task_rt로 next task로 뽑히면 곧 스케줄러에 의해 rq->curr(현재 실행중인 태스크)로 설정됨
  - exec_start를 rq_clock_task(현재 클럭 시간)으로 업데이트

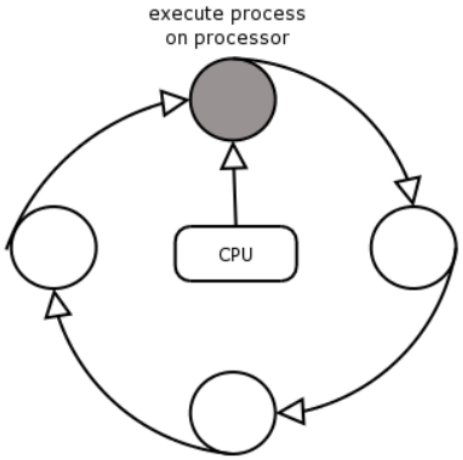
```
static struct task_struct *
pick_next_task_rt(struct rq *rq, struct task_struct *prev)
{
        put_prev_task(rq, prev);

        p = _pick_next_task_rt(rq);
}

static struct task_struct *_pick_next_task_rt(struct rq *rq)
{
        struct sched_rt_entity *rt_se;
        struct task_struct *p;
        struct rt_rq *rt_rq  = &rq->rt;

        do {
                rt_se = pick_next_rt_entity(rq, rt_rq);
        } while (rt_rq);

        p = rt_task_of(rt_se);
        p->se.exec_start = rq_clock_task(rq);

        return p;
}
```

<_pick_next_task_rt>

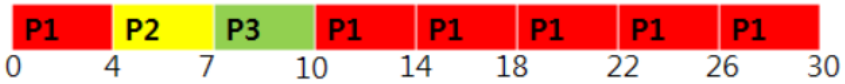
C. MYRR Scheduler

Round-Robin 스케줄러

➢ Round-Robin 스케줄러



프로세스	버스트 시간
P1	24
P2	3
P3	3



- 라운드 로빈의 뜻은 사람의 이름을 순서대로 적는 것이 아니라 원형으로 적어 조직의 서열을 숨기는 서명 방식을 말함
- 라운드 로빈은 시분할 시스템을 위해 설계된 알고리즘
- 시간 할당량 (time quantum) 또는 시간 조각 (time slice)만큼 task에게 cpu를 할당하는 스케줄링 알고리즘
- 리눅스의 RT 스케줄러의 SCHED_RR 방식이 라운드 로빈 사용

myrr 스케줄러 및 자료구조

- **myrr 스케줄러 특징**
  - **실행 큐(run queue): 리눅스가 제공하는 linked list를 사용하여 구현**
  - **스케줄러 우선순위: rt>mysched>myrr>cfs>idle**
  - **스케줄링 정책: Time Slice를 정확한 값으로 사용하기 어렵기 때문에 update_curr_rr 함수를 네 번 실행하면 스케줄링**

<include/linux/sched.h>

```
…
struct sched_myrr_entity {
        struct list_head  run_list;
        unsigned int   update_num;
};
…
struct task_struct{
        …
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
        struct sched_mysched_entity mysched;
        struct sched_myrr_entity myrr;
        …
};
…
```

<kernel/sched/sched.h>

```
…
struct myrr_rq {
        struct list_head  queue;
        unsigned int   nr_running;
};
…
 struct rq {
        …
        struct cfs_rq cfs;
        struct rt_rq rt;
        struct dl_rq dl;
        struct mysched_rq mysched;
        struct myrr_rq myrr;
        …
 }
 …
```

myrr.c

<kernel/sched/sched.h>

```
…
extern const struct sched_class idle_sched_class;
extern const struct sched_class mysched_sched_class;
extern const struct sched_class myrr_sched_class;
…
extern void init_dl_rq(structdl_rq*dl_rq, structrq*rq);
extern void init_mysched_rq(struct mysched_rq *mysched_rq);
extern void init_myrr_rq(struct myrr_rq *myrr_rq);
…
```

- **myrr.c를 다운로드하여 /usr/src/linux-3.18.27/kernel/sched에 삽입**
- **newclass2.c 파일을 다운받아 기존 newclass2.c 대체**

스케줄러 우선순위 변경

- 기존 스케줄러 우선순위: rt>fair>mysched>idle
- 변경 스케줄러 우선순위: rt>mysched>myrr>fair>idle

<kernel/sched/rt.c>

```
…
const struct sched_class rt_sched_class = {
        .next = &mysched_sched_class,
        …
}
```

<kernel/sched/mysched.c>

```
…
const struct sched_class mysched_sched_class = {
        .next = &myrr_sched_class,
        …
}
```

<kernel/sched/myrr.c>

```
…
const struct sched_class myrr_sched_class = {
        .next = &fair_sched_class,
        …
}
```

<kernel/sched/fair.c>

```
…
const struct sched_class fair_sched_class = {
        .next = &idle_sched_class,
        …
}
```

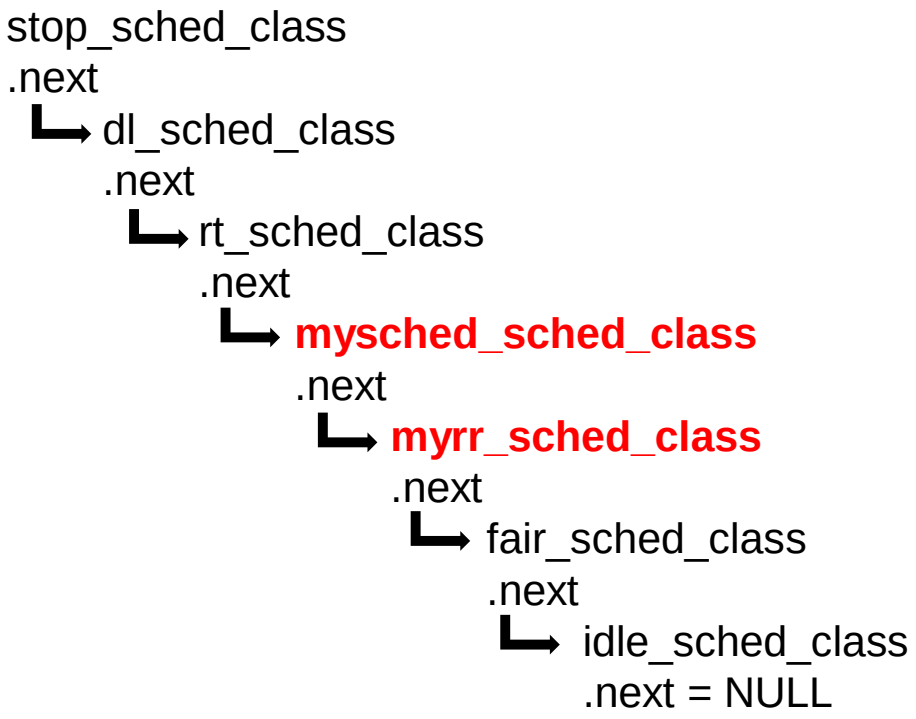
New Scheduler Class 추가

<kernel/sched/Makefile>

```
…
obj-y += core.o loadavg.o clock.o cputime.o
obj-y += idle_task.o fair.o rt.o deadline.o stop_task.o mysched.o myrr.o
```

<kernel/sched/core.c>

```
static void __sched_fork(unsigned long clone_flags, struct
task_struct *p)
{
        …
        INIT_LIST_HEAD(&p->mysched.run_list);
        INIT_LIST_HEAD(&p->myrr.run_list);
        …
}
void __initsched_init(void)
{
        …
        init_mysched_rq(&rq->mysched);
        init_myrr_rq(&rq->myrr);
        …
}
```



SCHED_MYSCHED policy

<kernel/sched/core.c>

```
static void __setscheduler(struct rq *rq, struct task_struct *p, const
struct sched_attr *attr, bool keep_boost)
{
        else if (mysched_policy(attr->sched_policy))
                p->sched_class= &mysched_sched_class;
        else if(myrr_policy(attr->sched_policy))
                p->sched_class=&myrr_sched_class;
        else
                p->sched_class= &fair_sched_class;
}
```

<include/uapi/linux/sched.h>

```
…
#define SCHED_DEADLINE      6
#define SCHED_MYSCHED       7
#define SCHED_MYRR          8
```

<kernel/sched/sched.h>

```
…
static inline int myrr_policy(int policy)
{
        return policy == SCHED_MYRR;
}
…
```

<kernel/sched/core.c>

```
static int __sched_setscheduler(struct task_struct *p, const struct sched_attr *attr, bool user)
{
        …
        if (policy != SCHED_DEADLINE &&
                policy != SCHED_FIFO && policy != SCHED_RR &&
                policy != SCHED_NORMAL && policy != SCHED_BATCH &&
                policy != SCHED_IDLE && policy != SCHED_MYSCHED && policy!= SCHED_MYRR)
        return -EINVAL;
        …
}
```

sched_setattr call tree

- sched_setattr
  - sys_sched_setattr
    - sched_setattr
      - __sched_setscheduler()
        - __setscheduler()

myrr 구현 및 실행

- 구현 방법
  - update_num: update_curr 함수가 실행된 횟수
  - update_curr_myrr 함수가 실행된 횟수가 4보다 큰 경우 스케줄링 실행
- kill $(pidof newclass2)를 통해 myrr 태스크를 kill 할 수 있음

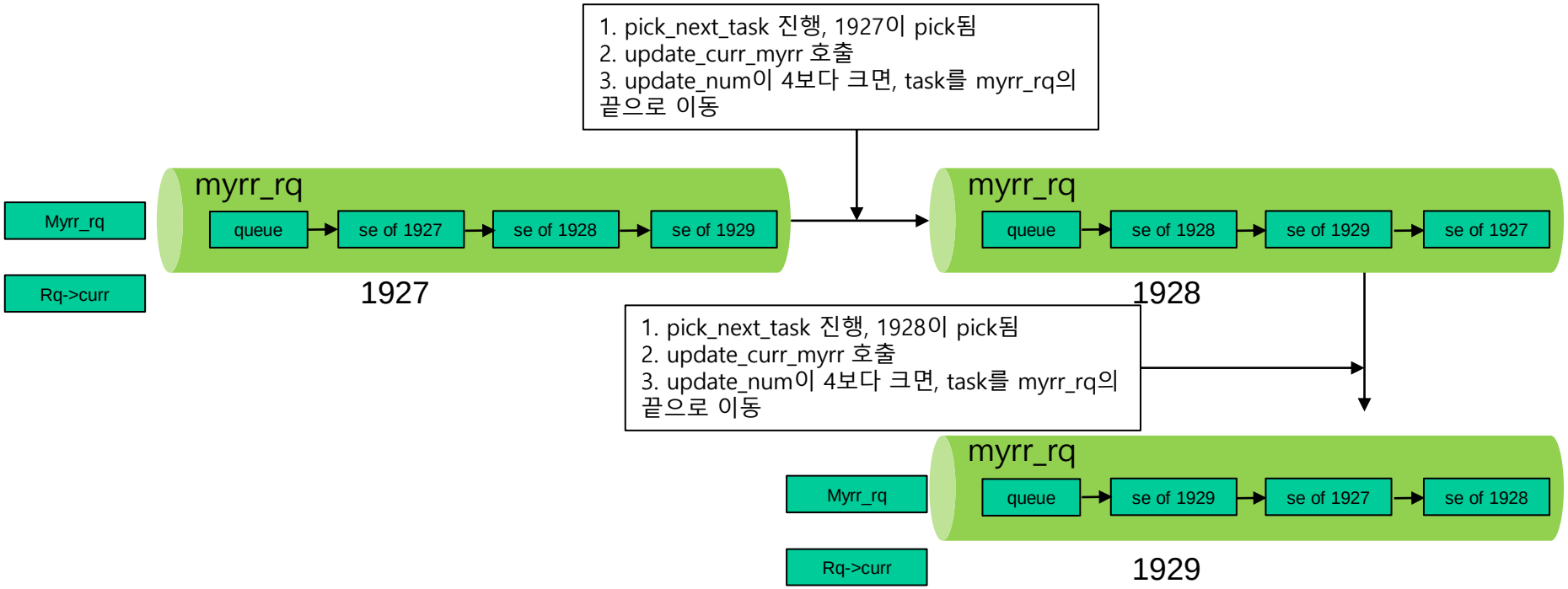
```
root@2019os:~/newclass# ./newclass r
cpuset at [0th] cpu in parent process(pid=2105) is succeed
Child's PID = 2106
Child's PID = 2107
Child's PID = 2108
Child's PID = 2109
forking 4 tasks is completed
***[NEWCLASS] Select myrr scheduling class
***[NEWCLASS] Select myrr scheduling class
***[NEWCLASS] Select myrr scheduling class
***[NEWCLASS] Select myrr scheduling class
root@2019os:~/newclass# cpuset at [1st] cpu in child proce
cpuset at [1st] cpu in child process(pid=2108) is succeed
cpuset at [1st] cpu in child process(pid=2107) is succeed
cpuset at [1st] cpu in child process(pid=2106) is succeed
pid=2109         j=0    result=100000000
pid=2107         j=0    result=100000000
pid=2108         j=0    result=100000000
pid=2106         j=0    result=100000000
pid=2109         j=1    result=100000000
pid=2107         j=1    result=100000000
pid=2108         j=1    result=100000000
pid=2106         j=1    result=100000000
pid=2109         j=2    result=100000000
```

```
  PID TTY          TIME CMD
 1994 pts/0    00:00:00 su
 1995 pts/0    00:00:00 bash
 2119 pts/0    00:00:00 newclass
 2120 pts/0    00:00:00 newclass
 2121 pts/0    00:00:00 newclass
 2122 pts/0    00:00:00 newclass
 2123 pts/0    00:00:00 ps
```

myrr 동작 방식

Example

- 1927,1928,1929의 pid를 가지는 myrr 프로세스들이 newclass 코드에 의해 1초 간격으로 fork 됨(mysched와 동일한 방식)
- Fork 된 각각의 프로세스는 1씩 200000000번 더하는 작업을 10번 반복
- 모든 작업이 끝난 뒤 프로세스 종료
- Update_num이 4보다 크면 reschedule 함수를 호출하여 스케줄링 실행



출력 결과(1)

```
[    0.000000] ***[MYRR] Mysched class is online
[    0.000000] ***[MYRR] Mysched class is online
[    0.000000] ***[MYRR] Mysched class is online
[    0.000000] ***[MYRR] Mysched class is online
[  130.862056] ***[MYRR] set_curr_task_myrr
[  130.862056] ***[MYRR] enqueue: success cpu=0, nr_running=1, pid=2109
[  130.862056]  ***[MYRR] pick_next_task: cpu=0, prev->pid=2109,next_p->pid=2109,nr_running=1
[  130.862056]  ***[MYRR] put_prev_task: do_nothing, p->pid=2109
[  130.862056]  ***[MYRR] dequeue: success cpu=0, nr_running=0, pid=2109
[  130.862056] ***[MYRR] enqueue: success cpu=1, nr_running=1, pid=2109
[  130.862056] ***[MYRR] set_curr_task_myrr
[  130.862056] ***[MYRR] enqueue: success cpu=0, nr_running=1, pid=2108
[  130.862056]  ***[MYRR] pick_next_task: cpu=0, prev->pid=2108,next_p->pid=2108,nr_running=1
[  130.862056]  ***[MYRR] put_prev_task: do_nothing, p->pid=2108
[  130.862056]  ***[MYRR] dequeue: success cpu=0, nr_running=0, pid=2108
[  130.862056] ***[MYRR] enqueue: success cpu=1, nr_running=2, pid=2108
[  130.862056] ***[MYRR] set_curr_task_myrr
[  130.862056] ***[MYRR] enqueue: success cpu=0, nr_running=1, pid=2107
[  130.862056]  ***[MYRR] pick_next_task: cpu=0, prev->pid=2107,next_p->pid=2107,nr_running=1
[  130.862056]  ***[MYRR] put_prev_task: do_nothing, p->pid=2107
[  130.862056]  ***[MYRR] dequeue: success cpu=0, nr_running=0, pid=2107
[  130.862056] ***[MYRR] enqueue: success cpu=1, nr_running=3, pid=2107
[  130.862056] ***[MYRR] set_curr_task_myrr
[  130.862056] ***[MYRR] enqueue: success cpu=0, nr_running=1, pid=2106
[  130.862056]  ***[MYRR] pick_next_task: cpu=0, prev->pid=2106,next_p->pid=2106,nr_running=1
[  130.862056]  ***[MYRR] put_prev_task: do_nothing, p->pid=2106
[  130.862056]  ***[MYRR] dequeue: success cpu=0, nr_running=0, pid=2106
[  130.862056] ***[MYRR] enqueue: success cpu=1, nr_running=4, pid=2106
[  131.596470]  ***[MYRR] pick_next_task: cpu=1, prev->pid=0,next_p->pid=2109,nr_running=4
[  131.596470]  ***[MYRR] dequeue: success cpu=1, nr_running=3, pid=2109
```

출력 결과(Update_num이 3보다 클 경우 예시)

```
[  153.781666]  ***[MYRR] pick_next_task: cpu=1, prev->pid=2109,next_p->pid=2106,nr_running=4
[  153.782653] ***[MYRR] update_curr_myrr        pid=2106 update_num=1
[  153.783643] ***[MYRR] update_curr_myrr        pid=2106 update_num=2
[  153.784633] ***[MYRR] update_curr_myrr        pid=2106 update_num=3
[  153.785624] ***[MYRR] update_curr_myrr        pid=2106 update_num=4
[  153.785624]  ***[MYRR] pick_next_task: cpu=1, prev->pid=2106,next_p->pid=2108,nr_running=4
[  153.786613] ***[MYRR] update_curr_myrr        pid=2108 update_num=1
[  153.787605] ***[MYRR] update_curr_myrr        pid=2108 update_num=2
[  153.788589] ***[MYRR] update_curr_myrr        pid=2108 update_num=3
[  153.789589] ***[MYRR] update_curr_myrr        pid=2108 update_num=4
[  153.789589]  ***[MYRR] pick_next_task: cpu=1, prev->pid=2108,next_p->pid=2107,nr_running=4
[  153.790582] ***[MYRR] update_curr_myrr        pid=2107 update_num=1
[  153.791667] ***[MYRR] update_curr_myrr        pid=2107 update_num=2
[  153.792574] ***[MYRR] update_curr_myrr        pid=2107 update_num=3
[  153.793563] ***[MYRR] update_curr_myrr        pid=2107 update_num=4
[  153.793563]  ***[MYRR] pick_next_task: cpu=1, prev->pid=2107,next_p->pid=2109,nr_running=4
[  153.794542] ***[MYRR] update_curr_myrr        pid=2109 update_num=1
[  153.795547] ***[MYRR] update_curr_myrr        pid=2109 update_num=2
[  153.796517] ***[MYRR] update_curr_myrr        pid=2109 update_num=3
[  153.797509] ***[MYRR] update_curr_myrr        pid=2109 update_num=4
[  153.797509]  ***[MYRR] pick_next_task: cpu=1, prev->pid=2109,next_p->pid=2106,nr_running=4
[  153.798497] ***[MYRR] update_curr_myrr        pid=2106 update_num=1
[  153.799489] ***[MYRR] update_curr_myrr        pid=2106 update_num=2
[  153.800487] ***[MYRR] update_curr_myrr        pid=2106 update_num=3
[  153.801478] ***[MYRR] update_curr_myrr        pid=2106 update_num=4
[  153.801478]  ***[MYRR] pick_next_task: cpu=1, prev->pid=2106,next_p->pid=2108,nr_running=4
[  153.802489] ***[MYRR] update_curr_myrr        pid=2108 update_num=1
[  153.803636] ***[MYRR] update_curr_myrr        pid=2108 update_num=2
[  153.804798] ***[MYRR] update_curr_myrr        pid=2108 update_num=3
[  153.805808] ***[MYRR] update_curr_myrr        pid=2108 update_num=4
[  153.805808]  ***[MYRR] pick_next_task: cpu=1, prev->pid=2108,next_p->pid=2107,nr_running=4
```

HW#9

제출

- **레포트(*.DOC) 작성**
  - 리눅스 스케줄러 전체 동작과 myrr 스케줄러 동작에 대한 설명
  - myrr.c 파일에 대한 설명(스크린샷 첨부)
    - ✓ 자료구조, 함수에 대해 설명
  - 최종 결과(dmesg 결과)에 대한 설명(스크린샷 첨부)
- **가상머신 제출**
  - 가상머신 내보내기를 사용하여 완성된 myrr이 컴파일 되어있는 ova 파일을 만들어서 본인 구글 드라이브에 올리고 해당 링크를 보고서에 추가
- **소스코드 제출**
  - myrr.c 파일을 보고서와 함께 zip파일로 묶어서 제출

기한 및 제출 장소

- **2021년 5월 28일 23시 59분까지**