Report for OS project 2

516030910125 Yining Liu

1. /arch/arm/configs/goldfish_armv7_defconfig

Add “CONFIG_WRR_GROUP_SCHED=y” to this file.

2. /include/linux/sched.h

a) Define `#define SCHED_WRR 6` so that the variable **policy** represents WRR mode when it`s 6, improving the readability of the code.

b) Define `struct sched_wrr_entity` referring to **sched_rt_entity** for WRR use. Add variable `unsigned int weight` in WRR.

c) Define `#define WRR_TIMESLICE (100 * HZ / 10000)` as 10ms timeslice for WRR. Different weights will be set for foreground and background apps later in **wrr.c** to get different timeslices.

d) Add variable `struct sched_wrr_entity wrr` in **task_struct**.

e) Declare `struct wrr_rq` to use it in this head file.

3. /kernel/sched/sched.h

a) Define functions **wrr_policy()** and **task_has_wrr_policy()** referring to corresponding **rt** functions.

```
static inline int wrr_policy(int policy)
{
    if (policy == SCHED_WRR)
        return 1;
    return 0;
}

static inline int task_has_wrr_policy(struct task_struct *p)
{
    return wrr_policy(p->policy);
}
```

b) Define a new struct `struct wrr_prio_array` referring to **rt**.

```
struct wrr_prio_array {
    DECLARE_BITMAP(bitmap, MAX_RT_PRIO + 1); /* include 1 bit for delimiter */
    struct list_head queue[MAX_RT_PRIO];
};
```

c) Declare a new struct `struct wrr_rq` .

d) Define the member of this struct **(wrr_rq)** referring to **rt_rq**.

c) Add variables `struct wrr_rq wrr` and `struct list_head leaf_wrr_rq_list` in struct **rq**.

d) Add variables `struct sched_wrr_entity **wrr_se` and `struct wrr_rq **wrr_rq` in struct **task_group**.

e) Declare some extern variables and functions for later use. These functions are later difined in **wrr.c**, including:

```
extern void free_wrr_sched_group(struct task_group *tg);
extern int alloc_wrr_sched_group(struct task_group *tg, struct task_group *parent);
extern void init_tg_wrr_entry(struct task_group *tg, struct wrr_rq *wrr_rq, struct sched_wrr_entity *wrr_se, int cpu, struct sched_wrr_entity *parent);
extern const struct sched_class wrr_sched_class;
extern void init_sched_wrr_class(void);
extern void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq);
```

4. /kernel/sched/core.c

a) Add `INIT_LIST_HEAD(&p->wrr.run_list);` in function **__sched_fork().**

b) Revise function **__setscheduler()** with an addtion of **wrr_sche_class:**

```
    if(p->policy==SCHED_WRR)
        p->sched_class = &wrr_sched_class;
    else if (rt_prio(p->prio))
        p->sched_class = &rt_sched_class;
    else
        p->sched_class = &fair_sched_class;
```

c) Revise function **__sched_setscheduler()** to enable **SCHED_WRR** :

```
    if (policy != SCHED_FIFO && policy != SCHED_RR &&
                policy != SCHED_NORMAL && policy != SCHED_BATCH &&
                policy != SCHED_IDLE && policy != SCHED_WRR)
            return -EINVAL;

if (param->sched_priority < 0 ||
    (p->mm && param->sched_priority > MAX_USER_RT_PRIO-1) ||
    (!p->mm && param->sched_priority > MAX_RT_PRIO-1))
    return -EINVAL;
if ((policy != 6) && (rt_policy(policy) != (param->sched_priority != 0)))
    return -EINVAL;
if (policy == 6 && param->sched_priority == 0)
    return -EINVAL;
```

d) Add `init_wrr_rq(&rq->wrr, rq);` in function **init_wrr_rq().**

e) Add `free_wrr_sched_group(tg);` in function **free_sched_group(struct task_group *tg).**

f) Declare extern function to get foreground/background information.

```
extern char *task_group_path(struct task_group *tg);
```

g) Revise function `int normal_prio` , taking **wrr** into account.

```
static inline int normal_prio(struct task_struct *p)
{
    int prio;

    if (task_has_rt_policy(p))
        prio = MAX_RT_PRIO-1 - p->rt_priority;
    else if (task_has_wrr_policy(p))
        prio = p->rt_priority;
    else
        prio = __normal_prio(p);

    return prio;
}
```

h) Revise function `void rt_mutex_setprio` , taking **wrr** into account.

```
if (rt_prio(prio) && p->policy != SCHED_WRR)
        p->sched_class = &rt_sched_class;
    else if(p->policy == SCHED_WRR)
        p->sched_class = &wrr_sched_class;
    else
        p->sched_class = &fair_sched_class;
```

5./kernel/sched/debug.c

Delete reserved word **static** in `char group_path[PATH_MAX]` and function `char *task_group_path(struct task_group *tg)` , which therefore can be used in **wrr.c** to differentiate foreground apps from background apps.

6./kernel/sched/rt.c

Revise **.next**, the member of `const struct sched_class rt_sched_class`, to make a **sched_class** list, or wrr cannot be called.

```
.next          = &wrr_sched_class,
```

7./kernel/sched/wrr.c

The file is written basically referring to **rt.c.** Like that in **rt.c**, we define a **basic timeslice**, only change it to 10ms. To meet requirements of this project , set the weight of a foreground task as 10 and background task as 1. Then use **weight * basic timeslice** as the actual timeslice for certain task.

In `static void enqueue_task_wrr(struct rq *rq, struct task_struct *p, int flags`, use function **task_group_path()** to judge if it is a foreground or background app, and assign corresponding value to **weight** in member **wrr_rq** in **p**. Then it calls **enqueue_wrr_entity()** to push the **wrr_se** of **p** into **rq.**

In `static void enqueue_wrr_entity(struct rq *rq, struct sched_wrr_entity *wrr_se, bool head)` , push **wrr_se** into the list of **rq**.

In `static void dequeue_task_wrr(struct rq *rq, struct task_struct *p, int flags)` (which calls `static void dequeue_wrr_entity(struct rq *rq, struct sched_wrr_entity`

`*wrr_se)`), the **wrr_se** of **p** is poped from the list of **rq** after the task is finished.

In `static void requeue_task_wrr(struct rq *rq, struct task_struct *p, int head)` ,the **wrr_se** of **p** is requeued when a timeslice ends and the task remains unfinished.

`static unsigned int get_rr_interval_wrr(struct rq *rq, struct task_struct *task)` returns the timeslice for current task according to its group information.

`static void task_tick_wrr(struct rq *rq, struct task_struct *p, int queued)` is called by function **scheduler_tick()** in **core.c**. Parameter **weight** is used to counts down each time this function is called, and therefore reduce the time left in current round for this task.