R06922134 葉沛陽 rts2017 project2

Correct Result

```
db@db-VirtualBox:~$ cd linux-2.6.32.60/
db@db-VirtualBox:~/linux-2.6.32.60$ cd test_weighted_rr/
db@db-VirtualBox:~/linux-2.6.32.60/test_weighted_rr$ gcc test_weighted_rr.c -lpt
hread -o test_weighted_rr
db@db-VirtualBox:~/linux-2.6.32.60/test_weighted_rr$ ./test_weighted_rr weighted
_rr 10 5 500000000
sched_policy: 6, quantum: 10, num_threads: 5, buffer_size: 500000000
abcdeabcdeabcdeabcdabcdabcabcabcabcababababababababababa
db@db-VirtualBox:~/linux-2.6.32.60/test_weighted_rr$
```

Implementation of the Five incomplete functions in "sched_weighted_rr.c"

1.

enqueuer_task_weighted_rr:用來把新的 process(p->weighted_rr_list_item)加進 run queue(rq->weighted_rr.queue)裡面。

list_add_tail 用來將新的 process 加進 run queue 。

*不能加進 rq 而是要加進 rq->weighted_rr.queue 因為他才是真正的 run queue for weighted_rr scheduler。

再來 nr_running 代表目前在 run queue 裡的 process 數量

所以要 rq->weighted_rr.nr_running++;。

```
static void enqueue_task_weighted_rr(struct rq *rq, struct task_struct *p, int wakeup, bool b)
{
    // not yet implemented
    list_add_tail(&(p->weighted_rr_list_item), &(rq->weighted_rr.queue));
    rq->weighted_rr.nr_running++;
    // ...
}
```

2.

Dequeuer_task_weighted_rr:用來把 process 移出 run queue

要先 update_curr_weighted_rr()來把當前的 task 的 run time 統計，如果這個 task 不是再 weighted_rr 裡的就 skip。

list_del 是能把 list 裡的指定的 item 刪除，所以用來刪掉想 dequeue 的 task

然後 nr_running 要減一。

```
static void dequeue_task_weighted_rr(struct rq *rq, struct task_struct *p, int sleep)
{
    // first update the task's runtime statistics
    update_curr_weighted_rr(rq);
    // not yet implemented
    list_del(&(p->weighted_rr_list_item));
    rq->weighted_rr.nr_running--;
    // ...
}
```

3.

yield_task_weighted_rr:就是當 task 想要放棄 running 退到 waiting 階段時要呼叫的。

requeue_task_weighted_rr:將 task 移到 run list 的最後面，就跟從 running 退到

waiting 階段一樣，所以呼叫這個 function 就可以了。

```
yield_task_weighted_rr(struct rq *rq)
{
    // not yet implemented
    requeue_task_weighted_rr(rq, rq->curr);
    // ...
}
```

4.

*pick_next_task_weighted_rr:用來選擇下一個應該 run 的 task。
首先要用 list_empty 確認 list 裡是否空的，如果是空的就 return NULL
list_first_entry:可以取出 list 中的第一個 entry 就是 next task。
因為這邊已經把舊的 requeue 或是 dequeue 所以這裡就不用再做一次了，如果再做一次就會有錯誤結果。
原本沒有注意到要加 next->se.exec_start = rq->clock;這句而發生錯誤。
後來看到下面的 put_prev_task_weighted_rr function 發現要設定 next 的起始執行時間設為目前的 clock。

```
static struct task_struct *pick_next_task_weighted_rr(struct rq *rq)
{
    struct task_struct *next;
    struct list_head *queue;
    struct weighted_rr_rq *weighted_rr_rq;


    // not yet implemented
    queue = &(rq->weighted_rr).queue;
    if (list_empty(queue))
        return NULL;
    // ...
    next = list_first_entry(queue, struct task_struct, weighted_rr_list_item);
    next->se.exec_start = rq->clock;
    /* you need to return the selected task here */
    //return NULL;
    return next;
}
```

5.

task_tick_weighted_rr 這個 function 是給 periodic scheduler call 的，每一段時間就會一次。
一開始，因為 pdf 沒有解釋這個 function 確切要做什麼事，所以不太清楚怎麼寫。
後來問了助教，知道了這個 function 會檢查 task_time_slice 是否用玩而且 task_time_slice 要減一，如果還沒用完的話就 return，用完的話就會輪到下一個 task，然後原本的 task 的 task_time_slice 要補充 weighted_time_slice 並 requeue，移到 queue 的最後面。

```c
static void task_tick_weighted_rr(struct rq *rq, struct task_struct *p,int queued)
{
    struct task_struct *curr;
    struct weighted_rr_rq *weighted_rr_rq;

    // first update the task's runtime statistics
    update_curr_weighted_rr(rq);

    // not yet implemented

    p->task_time_slice = p->task_time_slice - 1;
    if (p->task_time_slice){
        return;
    }

    p->task_time_slice = p->weighted_time_slice;


    requeue_task_weighted_rr(rq,p);
    // ...

    return;
}
```

這邊原本我多寫了 pick_next_task_weighted_rr 想說要換下一個 task，結果發生下圖的錯誤，因為多了 pick_next_task_weighted_rr 造成 a 還沒 run 就換到 b 所以產生了 abcdeabcdeb 這樣的結果。

```
db@db-VirtualBox:~/linux-2.6.32.60$ cd test_weighted_rr/
db@db-VirtualBox:~/linux-2.6.32.60/test_weighted_rr$ gcc test_weighted_rr.c -lpt
hread -o test_weighted_rr
db@db-VirtualBox:~/linux-2.6.32.60/test_weighted_rr$ ./test_weighted_rr weighted
_rr 10 5 500000000
sched_policy: 6, quantum: 10, num_threads: 5, buffer_size: 500000000
abcdeabcdeb
db@db-VirtualBox:~/linux-2.6.32.60/test_weighted_rr$
```

強大又好用的 list_head 結構

因為 pdf 裡的連結壞了，我又找了另一個網頁也是在介紹 list_head。

https://myao0730.blogspot.tw/2016/12/linux.html

裡面有很多常用的 list_head functions，在這次作業用到了很多。

list_head 只包含 2 個指標， 分別用來指向前一個(prev)以及後一個(next)的節點位址。

以下舉一些常用的 function:

__list_add:可以將 item 加入 list 指定的位置。

list_add:將 item 加入 list 的頭。

list_add_tail:將 item 加入 list 的尾。

__list_del :可以刪除某個 entry，但需要先知道這個 entry 的前/後端節點。

list_entry(ptr,type,member) :可以用來得知 struct 中某一 member 的位址相對於該 struct 起始位址的距離。

list_for_each(pos, head): 走訪整個鏈結串列，pos 為一個指標並指向第一個項目(head->next)，以此為起點一路訪問下去，終止條件是當 pos

指向 head 時，代表已經走訪完一圈了。

Other Errors

這次作業除了 code 的部分，在建立 kernel 以及因為這次是寫 kernel code 所以每次修改完 code 都必須重新編 kernel，然後也遇到了其他問題。
一開始發生了如下圖 kernel build 失敗的問題，後來請問了助教，才發現要把 CPU 改成 2 顆以上，而且如果 code 寫錯了的話，有時候是結果會錯，但有的時候也會造成 kernel build 失敗 ，就要再用乾淨的重新 build 一次，真的很花時間。