B07902080 黃義峰

1.設計

對於四種不同 policy 的情況，我寫了不同的處理函數。

(1)FIFO:因為 sched_setscheduler 中有 sched_FIFO 可用，我也寄信問過助教說可以使用，所以處理 FIFO 的情況我是根據任務的加入順序把每個任務 set 成 sched_FIFO，且 param.sched_priority 設成相同。因為在 sched_FIFO 的情況下，會先按照 param.sched_priority 從大到小運行，如果 priority 相同，則按照任務加入的順序運行，我這裡只運用了後半部分。

(2)PSJF:這個部分我寫了一個 link list 來記錄當前時間，所有已經加入的任務的狀況，並按照每個任務的剩餘時間進行排序，剩的越少的排在越前，已經結束的任務剩餘時間是 0 所以結束之後其位置不會動。每當有新任務加入後，就把新任務的 status 也加入 link list(因為 PSJF 的任務 priority 只會在新任務加入時發生改變，這點 FIFO,SJF 也是一樣)。而當處理完新任務的加入 link list 後(這也是在 main 中操作)，在 main process 中我把所有任務按 link list 中的順序再重新賦予每個任務一個新的 priority，同樣我這裡也是用的 sched_FIFO,因為當 priority 更大的任務出現時，當前運行的 process 也會讓出 cpu。因此此時每個任務的 priority 大小是不同的。

(3)RR:RR 的 policy 與其他的有一點不一樣，其他的 policy 中當全部任務加入結束時，每個任務的 priority 是固定好的，也可以說通過一開始知曉全部的 process 的 ready 和 burst 我們就可以排出他應該在的 priority，不過 RR 中 process 的 priority 是會根據你運行了的次數改變的(同時數個任務的情況)。所以我在 main process 中不斷地(以 100unit 為單位)去監控 process 的運行情況，如果測試到某個 child 剛剛運行完這次的 quantum，(同樣我這裡也用的是 sched_FIFO)，則會先把這個 child 的 priority 調低，然後馬上在調回和其他的 child 一樣的 priority。不過因為是 sched_FIFO，雖然 priority 相同，但是他此時在 queue 中的位置是最後，所以便能保持任務之間運行的輪轉。

(4)SJF:SJF 與 PSJF 的部分大同小異，也可以使用 link list 來記錄，不過唯一要注意的點是因為當前運行的 process 不能被打斷，所以新 process 添加時，insert 進 link list 時要注意，新的 process 的剩餘時間哪怕比第一個在運行的 process 短，他也必須要 insert 到其後面一位。不過在本次中，我嘗試使用了另一種方法。通過自己了一個簡單的演算法，計算每個任務在整體看下來中所有任務中該有的 process。之後每當新的 process 運行時直接把先前計算的 priority 賦予它，因此連續兩個的 process0 和 1 可能一上來得到的 priority 卻是 94 98(舉例)。我這裡也同樣用的是 sched_FIFO。

gettime.c

```c
#include <linux/linkage.h>
#include <linux/kernel.h>
#include <linux/ktime.h>
#include <linux/timekeeping.h>

asmlinkage long sys_gettime(struct timespec *time)
{
    getnstimeofday(time);
    return 0;
}
```

核心版本

showinfo.c

```c
#include <linux/linkage.h>
#include <linux/kernel.h>
#include <linux/time.h>

asmlinkage long sys_showinfo(struct timespec start,struct timespec finish,pid_t pid)
{
    printk("[Project1] %d %ld.%ld %ld.%ld\n",pid,start.tv_sec,start.tv_nsec,finish.tv_sec,finish.tv_nsec);
    return 0;
}
```

2.核心版本

通過 homework1 中的方法，我寫了兩個 system cal(上圖)l 分別是 gettime 和 showinfo。gettime 通過 syscall 可以引用 getnstimeofday，即在我的 code 中引用 gettime(&time),(實際上是 syscal(334,&time)),gettime 再引用 getnstimeofday(time),便能取得當前的時間，在每個 child 剛運行和結束時各呼叫一次。然後是 showinfo(start_time,finish_time,pid)，他會讀入之前兩次 gettime 的值和 child 的 pid,並將其按照格式 printk 出來。

3.差異

產生差異是意料之內的，實際上的 cpu 也會要處理別的狀況。我把 main process set 在 CPU0 上運行，而所有的 child 在 CPU1 中，使得 child 不會搶佔 main process 的運行。因為首先是在不同的 cpu 上，速度可能有一點差別，而且 main 和 child 除了 run unit time 的 function 外還會跑別的不同的東西，也會導致時間越長，後面兩者都 time_cnt 差距會變大一點。不過在測資中還體現不出來。而且 FIFO,SJF,PSJF 中是在 child 剛加入 ready 時它的 priority 就決定好了，所以只要不是新 child 的 ready 時間和上一個 child 結束的時間剛好重合，就不會有太大的問題。不過測資中有一個地方可以看到這種情況，那是 RR_3 的測資，其中 P5 的 reay 時間剛好這時候 P1 剛完成一個 quantum 這時兩邊 cpu 的時間的些許差距便能體現出來。我運行過好幾次，也重啟電腦試過，發現有的時候會是 child1 先結束重新回到 waiting queue 後才是 P5 的加入，而也有發生先 P5 加入然後 child1 才結束回來的情況。這樣會影響到一點運行的順序，從 dmesg 的結果也可以看出，這個測資結束的順序是 P3,P1,P2。但有時候 P3 和 P1，P1 和 P2 之間的時間差是 1500unit 和 500unit，而有時候又是 1000unit 和 1000unit，其中的差別就在上面所述的順序。

此外，因為 RR 的特性，對於 main 和 child 之間時間的同步就更為重要。一開始我也是分在兩個 cpu 上運行，但發現 main 總是比 child 要慢不少，導致不能很及時的切換到下一個 child 而產生順序錯誤。後來發現了在 set_scheduler 後會把那個 cpu 先拿來運行 set 過的實時進程任務，所以 cpu1 中的運行效率會比較高。而 main 在 cpu0 中我一開始沒有也將其 set_scheduler，所以 main 會慢一些。後來把 main 在一開始也設置成 sched_FIFO 後就解決兩邊同步的問題了。

```
[  897.758260] [Project1] 2574 1588093290.405955032 1588093321.362644902
[  899.937539] [Project1] 2572 1588093283.969271683 1588093323.541966548
[  902.160885] [Project1] 2573 1588093287.233430851 1588093325.765354663
[  918.381732] [Project1] 2577 1588093297.909426444 1588093341.986498658
[  922.594756] [Project1] 2576 1588093295.731195434 1588093346.199602483
[  924.709586] [Project1] 2575 1588093293.540832336 1588093348.314457515
```

```
[Project1] 2553 1588139935.755266398 1588139967.295432057
[Project1] 2551 1588139928.704582832 1588139970.800002051
[Project1] 2552 1588139932.174931216 1588139971.909926906
[Project1] 2556 1588139944.655768677 1588139990.285501955
[Project1] 2555 1588139941.319659185 1588139994.839128139
[Project1] 2554 1588139940.203647487 1588139997.2810013
```

可以看出我跑不同次時，因為 cpu 的狀況不同，前三者的結束時間差有些不同。這可能跟我的電腦狀況有關，上圖是昨天測的，下圖是今天開了一晚上測的，不知道有沒有很大影響。