OS project 1 report

b06902111 資工三 林慶珠

設計

我使用兩顆cpu，一顆scheduler，一顆專門跑process(job)。

以下的變數意義：`running` 表示現在跑的process的index，如果`running == -1` 表示沒人在跑。一個process做完或還沒ready `pid == -1` (沒工作要做)。

我用process的ready time當成start time，也就是行程的產生時間當成start time。

Scheduler

如果是FIFO, SJF, PSJF都是有新的人ready才跑algorithm決定誰要跑：

```
if( policy != RR && new_come == 1 ){
        //select one from algorithm, wake up next
        int next = next_process( proc, N, policy);
        //next == -1 : no job ready / all done
        if( next != -1 ){
                if( running != next ){
                        proc_setscheduler(proc[next].pid, SCHED_OTHER); //wakeup
                        proc_setscheduler(proc[running].pid, SCHED_IDLE); //block

                        if( running != -1 ) last_run = running;
                        running = next;
                        rr_time_cumulate = 0;
                }
        }
}
```

如果是RR就是如果現在沒人跑，或是某個人的time quatum用完了，才跑algorithm：

```
else if( running == -1 || rr_time_cumulate%RR_TIMEQUANTUM == 0 ){
        //select one from algorithm, wake up next
        int next = next_process( proc, N, policy);
        //next == -1 : no job ready / all done
        if( next != -1 ){
                if( running != next ){
                        proc_setscheduler(proc[next].pid, SCHED_OTHER); //wakeup
                        proc_setscheduler(proc[running].pid, SCHED_IDLE); //block

                        if( running != -1 ) last_run = running;
                        running = next;
                        rr_time_cumulate = 0;
                }
        }

}
```

FIFO

如果有人在跑就讓他繼續跑(non-preemptive)，跑完會在scheduler那邊設成`pid=-1`。因為一開始的時候proc是用ready time排序，所以直接iterate proc直接拿到下一個。

```
if( policy == FIFO ){
    if( running != -1 ) return running; //non-preemptive

    //proc is sorted by ready time
    for( int i = 0 ; i < N ; i++){
        if( proc[i].pid != -1 ) return i; //ready and has job
    }
    return -1; //no next, all job done
}
```

RR

如果有人在跑而且他在一個time quantum裡面就讓他繼續跑。

如果要換人有分兩種：上一個跑完了，此時`running == -1`會從`last_run`知道剛剛跑完的是誰，那麼就從proc裡面的下一個process開始找是否有工作要做。如果一個跑到一半時間到了，會從`running` index往後找誰有工作要做。

```
else if( policy == RR ){
    //run complete time quantum
    if( running != -1 && (rr_time_cumulate % RR_TIMEQUANTUM) != 0 ){
        if( rr_time_cumulate >= RR_TIMEQUANTUM ) rr_time_cumulate %= RR_TIMEQUANTUM;
        return running;
    }

    if( running == -1 ){ // pick new one from last
        for( int i = last_run+1; i < N+last_run+1 ; i++)
            if( proc[(i%N)].pid != -1 ) return (i%N);
    }
    else{ //time up pause
        for( int i = running+1 ; i < N+running+1 ; i++){
            if( (i%N) == running ) continue;
            if( proc[(i%N)].pid != -1 ) return (i%N);
        }
    }
    return -1; //no next, all job done
}
```

SJF 和 PSJF

SJF 和 PSJF 只有差在是不是preemptive而已。如果有剩下最少的就執行它，如果剩下最少的人跟`running`剩下的時間一樣就繼續執行running process（減少context switch）。

```
else{ //SJF and PSJF

        if( policy == SJF ){
                if( running != -1 ) return running; //non-preemptive
        }

        int min_idx = -1;
        for( int i = 0 ; i < N ; i++){
                if( proc[i].pid == -1 ) continue;

                if( min_idx == -1 ) min_idx = i;
                else if( proc[i].t_exec < proc[min_idx].t_exec ) min_idx = i;
        }

        if( min_idx == -1 ) return -1; //no job

        if( running != -1 && proc[min_idx].t_exec == proc[running].t_exec ) return running;
        else return min_idx; //less context switch
}
```

核心版本

Linux 4.14.25

比較實際結果與理論結果，並解釋造成差異的原因

一開始我把應該要印在kernel log的message印出來的時候，我發現跑process(job)的cpu會比scheduler的cpu早完成工作（順序對，會早一點點），這可能是因為scheduler不只有計算時間（需要計算時間來看誰ready），也有判斷現在現在該輪到誰來做，所以花的時間會比較久。

使用說明

執行之前要先照著hw1的投影片把`kernel_files`的檔案放到該放的位置並編譯kernel。

```
$ make
$ sudo ./main.out < [data in] > [data out]
```

參考資料

Linux man page

https://www.gnu.org/software/libc/manual/html_node/index.html#SEC_Contents

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/reference_guide/sect-using_library_calls_to_set_priority

https://www.princeton.edu/~cad/emsim/files/example.c