b07902040 資工二 吳承軒

OS Project1 report

1.設計

Overall

```
typedef struct pppp{
    char name[32];
    int readyt, exet;
    pid_t pid;
}Process;
```

將每個Process存成以上形式，對於尚未被scheduled到的，或是已經結束的process，設定pid為-1，並用ready_time進行sort，ready_time越早的放在越前面。

用一個while迴圈，維護in_cpu_process_pid、ready_cnt等數，每一輪做的事:

1.檢查是否有process已結束。

2.檢查下一個process在此時是否ready，若ready就fork一個child process，並block住。

3.根據scheduler policy判斷是否需要content switch，若需要應switch到哪個process。

4.run an unit time、my_time++、P[in_cpu].exet--=1，進入下一段時間。

標記已ready的process

由於已經先對ready_time進行sort，所以process會照index順序依序ready。考慮到ready_time可能會相同，而每個while迴圈都檢查一次所有process是否ready非常耗時，所以我做了一些設計，讓"標記已ready的process"的總時間複雜度在O(max(each ready_time, |Process|))，同時兼顧ready_time相同的情況。

實現控制Content Switch

使用sched_setscheduler，將模式設為FIFO，調降要block或要離開cpu process的priority；調升要進入cpu process的priority。

使用sched_setaffinity將main(parent) process限制在cpu0執行，child process限制在cpu1執行，避免系統自動使用多核運行，打亂想要的schedule。

Policy(對應3.)

先檢查in_cpu_process_pid是否為-1，-1表示沒有process在cpu中執行，非-1表示其值為在cpu中process 的pid

以下runable process表示已ready且尚未結束的process。

switch表示content switch。

- FIFO

  非-1 : 由於FIFO是non-preemptive，所以不能switch。

  -1 : switch到runable process中ready_time最早的。

- RR(old)

  非-1：檢查my_time-pre_time是否為Quantum的倍數，pre_time為上一次switch的時間。若是表示switch到下一個runable process；若不是則不能switch。

  -1：從pre_in_cpu開始，找到runable process就switch，pre_in_cpu是前一個在cpu內的process pid

- RR 使用queue，ready的就直接進queue尾端 非-1：檢查my_time-pre_time是否為Quantum的倍數，pre_time為上一次switch的時間。若是就將in_cpu push到queue尾端，再從頭pop出來 -1：若queue不為空就pop queue做為in_cpu

- SJF

  非-1：由於SJF是non-preemptive，所以不能switch。

  -1：switch到runable process中exet_time最短的。

- PSJF

  無論-1或非-1:switch到runable process中exet_time最短的。

2. Kernel version and Platform

Kernel : linux 4.14.25

Platform : Ubuntu 16.04LTS on Intel corei7-7700HQ @ 2.8Ghz

3. 比較實際結果與理論結果

以下以要求錄影的測資為例，借用林楷恩(b07902075)同學寫的測試code:

```
FIFO_1.txt:
    Process P1:
        theory:     start at 0, end at 500
        my_result:  start at 0, end at 503
        difference: start_time 0.0%, end_time +0.60%, run_time 3 units
    Process P2:
        theory:     start at 500, end at 1000
        my_result:  start at 503, end at 1018
        difference: start_time +0.60%, end_time +1.80%, run_time 15 units
    Process P3:
        theory:     start at 1000, end at 1500
        my_result:  start at 1018, end at 1519
        difference: start_time +1.80%, end_time +1.27%, run_time 1 units
    Process P4:
        theory:     start at 1500, end at 2000
        my_result:  start at 1519, end at 2015
        difference: start_time +1.27%, end_time +0.75%, run_time 4 units
    Process P5:
        theory:     start at 2000, end at 2500
        my_result:  start at 2015, end at 2491
        difference: start_time +0.75%, end_time -0.36%, run_time 24 units

    * The order of finish time is correct
    * Average run time difference of FIFO_1 = 9.4 units
```

```
PSJF_2.txt:
    Process P1:
        theory:     start at 0, end at 4000
```

```
        my_result:  start at 0, end at 3979
        difference: start_time 0.0%, end_time -0.53%, run_time 21 units
    Process P2:
        theory:     start at 1000, end at 2000
        my_result:  start at 976, end at 1981
        difference: start_time -2.40%, end_time -0.95%, run_time 5 units
    Process P3:
        theory:     start at 4000, end at 11000
        my_result:  start at 3979, end at 11012
        difference: start_time -0.53%, end_time +0.11%, run_time 33 units
    Process P4:
        theory:     start at 5000, end at 7000
        my_result:  start at 4931, end at 6876
        difference: start_time -1.38%, end_time -1.77%, run_time 55 units
    Process P5:
        theory:     start at 7000, end at 8000
        my_result:  start at 6896, end at 7903
        difference: start_time -1.49%, end_time -1.21%, run_time 7 units

    * The order of finish time is correct
    * Average run time difference of PSJF_2 = 24.2 units
```

```
RR_3.txt:
    Process P1:
        theory:     start at 1200, end at 19200
        my_result:  start at 1200, end at 19864
        difference: start_time +0.00%, end_time +3.46%, run_time 664 units
    Process P2:
        theory:     start at 2700, end at 20200
        my_result:  start at 2657, end at 20398
        difference: start_time -1.59%, end_time +0.98%, run_time 241 units
    Process P3:
        theory:     start at 4200, end at 18200
        my_result:  start at 4124, end at 17787
        difference: start_time -1.81%, end_time -2.27%, run_time 337 units
    Process P4:
        theory:     start at 6200, end at 31200
        my_result:  start at 6097, end at 31212
        difference: start_time -1.66%, end_time +0.04%, run_time 115 units
    Process P5:
        theory:     start at 7200, end at 30200
        my_result:  start at 6582, end at 29985
        difference: start_time -8.58%, end_time -0.71%, run_time 403 units
    Process P6:
        theory:     start at 8200, end at 28200
        my_result:  start at 7065, end at 27749
        difference: start_time -13.84%, end_time -1.60%, run_time 684 units

    * The order of finish time is correct
    * Average run time difference of RR_3 = 407.3333333333333 units
```

```
SJF_4.txt:
    Process P1:
        theory:     start at 0, end at 3000
        my_result:  start at 0, end at 3079
        difference: start_time 0.0%, end_time +2.63%, run_time 79 units
```

```
    Process P2:
        theory:     start at 3000, end at 4000
        my_result:  start at 3079, end at 4115
        difference: start_time +2.63%, end_time +2.88%, run_time 36 units
    Process P3:
        theory:     start at 4000, end at 8000
        my_result:  start at 4115, end at 8210
        difference: start_time +2.88%, end_time +2.62%, run_time 95 units
    Process P4:
        theory:     start at 9000, end at 11000
        my_result:  start at 8211, end at 11245
        difference: start_time -8.77%, end_time +2.23%, run_time 1034 units
    Process P5:
        theory:     start at 8000, end at 9000
        my_result:  start at 8211, end at 9205
        difference: start_time +2.64%, end_time +2.28%, run_time 6 units

    * The order of finish time is correct
    * Average run time difference of SJF_4 = 250.0 units
```

difference是理論時間和實際時間之間的差異，實際時間透過從TIME_MEASUREMENT中算出單位時間的時長，轉換成和理論時間相同的scalor。

我的start time紀錄的是在fork出來的child process，要跑第一次unit_time之前的時間，可以發現如SJF_4中，P4和P5的start time都為8211，這是因為P4和P5在當初ready時就已經被fork出來並block住，而往後一直都有較高priority的process在運行。在沒有較高的priority運行時(P3於8210結束)，就輪到P4和P5運行，同時output了start time。此時schduler就會找下一個應該被選中的，應該為P5，就將P5的priority提高，P4在跑到unit_time之前就又被block住。

所以P4真正的start time(開始跑unit_time)是正確的，從end_time和結束順序也能驗證這點。

造成差異的可能因素

- 當下的系統負荷，有些cases中實際時間早於理論時間，可能的原因是，相較於run某些cases時，run TIME_MEASUREMENT時系統負荷較高，導致計算出的單位時間較長。
- CPU的速度不完全是固定的。
- scheduler的執行本身就有cost，使實際時間較理論時間更慢一些，而FIFO的difference較其他policy小，推測也是因為FIFO的策略較為單純，scheduler花費時間較少。