OS Project 1

資工三 b06902016 林紹維

環境

OS: Ubuntu: 16.04
GCC version: 5.4.0
Virtual Hardware:

- 2G RAM
- 2 Processors (Host: i7-7500u, 2 cores, 4 threads)

設計

這次的 project 分成了三份程式碼， main.c、process_control.c 及 scheduler.c。需要注意的是，因為 spec 中並沒有強調列印到 stdout 的 *name pid* 是要在 process 跑完之後，或是在 process 第一次被丟上CPU的時候印出來，所以本次設計的方式是 process 在第一次被丟上CPU的時候就輸出，而 dmesg 則是在該 process 執行完畢後會輸出。另一件值得注意的事是，每個 process 在被 fork 之後都有 pipe 可以跟 scheduler 溝通，當scheduler 決定讓該 process 執行時，會透過 pipe 傳達訊息，而收到訊息的 child process 此時才會執行一個時間單位，這個溝通是在每個時間單位都會執行一次的。

main.c

main.c讀取了 input 之後，便會透過 policy 決定呼叫 scheduler 中相對應的 policy 的函式，完成工作的排程。

scheduler.c

此份程式碼中有主要的四大函式對應四種 policy， `scheduler_FIFO`, `scheduler_SJF`, `scheduler_PSJF` 及 `scheduler_RR` 。FIFO 的函式基本上就是根據ready time sort完之後，逐一執行到所有 process 都執行完畢。SJF及PSJF會有一個函式 `find_shortest` 來找到當下執行時間最短的 process。不同的是，SJF 在決定好該 process 執行後，就不會被中斷，直到該 process 執行完畢；而 PSJF 則是每執行一個時間單位，就會呼叫 `find_shortest` 來看現在是否有更短時間的 process 需要被優先執行。最後 RR 則是在每500個時間單位選擇下一個 process 的執行，這次實作的選取是透過 for 迴圈到下一個 index ，因此並不是透過 queue 來維持 RR 的順序的。

process_control.c

這份程式碼中，定義了如何做 context switch ，包括暫停一個 process `proc_kickout` 、延續已經開始的 process `proc_resume` 及如何 fork 並 create 一個 process `proc_create` 。

Makefile

- all : 編譯所有.c檔成.o檔，並產生main

核心版本

這次所使用的核心版本是 4.15.0 。

```
root@cnlab-VirtualBox:~/OS/my_os# uname -a
Linux cnlab-VirtualBox 4.15.0-45-generic #48~16.04.1-Ubuntu SMP Tue Jan 29 18:03
:48 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

實驗結果

下方是某次執行 TIME_MEASUREMENT 得到的 dmesg，透過程式計算出每 500 個 unit 所花的時間，平均後大約為 0.99 秒每 500 個 unit。

```
root@cnlab-VirtualBox:~/OS/my_os# dmesg | python3 check_file/check_time.py
[0.         0.9994154] 0.999415397644043
[1.90322375 2.87105513] 0.9678313732147217
[3.72206283 4.72780752] 1.0057446956634521
[5.65011501 6.65474296] 1.0046279430389404
[7.5240891  8.50386238] 0.9797732830047607
[ 9.42051411 10.46740675] 1.0468926429748535
[11.34559155 12.34726191] 1.0016703605651855
[13.24481559 14.26032925] 1.0155136585235596
[15.13973236 16.11459446] 0.9748620986938477
[17.04204035 18.04478025] 1.0027399063110352
```

以下透過表格來分析理論值與實際值的差別，其中透過要跑的單位/500再乘上 0.99 就會是理論的執行時間； 實作的時間因為是直接取dmesg上的數值，所以起始時間不會是 0 ，但透過相減得到執行時間，就可以做比較了。

- FIFO_4.txt

Process	Expected start time	Expected finish time	Expected execute time	Real start time	Real finish time	Real execute time
P1	0	3.960	3.960	40.639	44.647	4.008
P2	3.960	4.950	0.990	44.647	45.645	0.998
P3	4.950	5.346	0.396	45.645	46.049	0.404
P4	5.346	6.336	0.990	46.049	47.061	1.012

- PSJF_3.txt

Process	Expected start time	Expected finish time	Expected execute time	Real start time	Real finish time	Real execute time
P1	0	6.930	6.930	66.875	73.808	6.933
P2	0.990	1.980	0.990	67.898	68.875	0.977
P3	1.980	2.970	0.990	68.878	69.848	0.970
P4	2.970	3.960	0.990	69.849	70.892	1.043

- RR_2.txt

Process	Expected start time	Expected finish time	Expected execute time	Real start time	Real finish time	Real execute time
P1	1.188	16.038	14.850	57.998	72.919	14.921
P2	2.178	19.008	16.830	59.018	76.532	17.514

觀察上方三個表格可以發現，絕大部分的實際值都比理論值高了一些，原因在於理論值並沒有考慮到其他的硬體因素，又或是當下執行程式時，有其他的程式也要用CPU而發生了context switch，造成實際值所花的時間比較長。但透過數據的計算，實際值多了理論值的1.8%，為可接受的範圍，證明此次的實作中沒有太多的干擾，程式運行也算順利!