2020 OS Project 1 —— Process Scheduling

學號：B07902037 學生：蔡沛勳

Referenced：OS Group27

https://github.com/andy920262/OS2016/tree/master/project1

1. 設計：

A. 主要概念：使用兩顆 CPU，父程序用一顆 CPU 來排程並用另一顆 CPU 執行子程序。

B. 主要函式介紹：

(1) 主要概念：使用兩顆 CPU，父程序用一顆 CPU 來排程並用另一顆執行子程序。

(2) 程式邏輯：

main()：負責讀取輸入後啟動 schedule()進行排程。

schedule()：先設定好自己的 CPU 及優先權後，以 while 迴圈作為時間單位依序執行五個步驟。

a. 確認上一個時間使否有子程序執行完畢，有則 wait 它。

b. 確認是否有程序會 arrive，有則用 exec_process()執行並 block 它。

c. 根據排程方式找出該時間單位應該要執行的子程序。

d. 判斷是否已全部執行完畢 / 是否需要 context switch。

e. 跑一單位的時間並對執行子程序進行紀錄。

Context switch：用 block_process() 來降低被指定程序的優先權，用 wait_process() 來提高被指定程序的優先權。子程序中同時只會有一組處在 wake 的狀況，其餘皆被父程序 block 住。

find_next()、queue_push()及 queue_pop()：根據不同的排程方式找出當時該被執行的子程序。

exec_process()：用 fork()產生子程序後，父程序設定好 CPU，子程序進行六個步驟。

a. 輸出自己的名稱及 pid。
b. 用 syscall 的 get_time 得到開始時間。
c. 用 for 迴圈執行需要執行的單位時間。
d. 用 syscall 的 get_time 得到結束時間。
e. 用 syscall 的 p1_printk 將 pid、開始與結束時間輸出。
f. 用 exit 結束該子程序。

2. **核心版本：**Kernel 使用 linux-4.14.25 版

3. 實際結果與理論結果比較

A. 實際運行時的同步問題：

若使父程序及子程序在每單位時間結束時產生一次輸出，理想中父程序及子程序的輸出應該交錯排列。然而實際上常常發生在父程序執行一單位時間的其間子程序已經執行超過一單位時間的狀況。猜想原因為父程序在單位時間以外還要花時間進行排程相關的處理導致耗時較長，或是有其他程序在跟父程序爭取同一顆 CPU。此錯誤嚴重時可能造成因子程序提早 exit 因此父程序無法 wake 該子程序導致的錯誤，亦或是程序結束的順序不對等等。

可行的解決方法為每此子程序跑完一單位時間就進行 block，而 parent 每單位時間再重新 wake 該子程序使其繼續運行。但缺點為浪費資源在不必要的 context switch 上。

B. 單位時間在執行上的差異：

以 FIFO_1 及 TIME_MEASUREMENT 為例。

(計算方式： FIFO_1：子程序結束時間 – 上一個子程序結束時間

TIME_MEASUREMENT：子程序結束時間 - 子程序開始時間)

跑 500 次 unit_of_time() 所需時間 (單位:秒)		
Process name	FIFO_1	TIME_MEASUREMENT
P0	X	0.8580918312
P1	1.46710038185	0.83621668815
P2	1.23223948479	0.87341070175
P3	1.06281757355	0.87829637527
P4	1.25473427773	0.88128948211
P5	1.08985877037	0.87623047828
P6	X	0.87061142921
P7	X	0.87830710411
P8	X	0.88336873054
P9	X	0.84935641288
平均	1.22135009766	0.86851792335

可根據上述結果觀察雖然各自內的誤差較小，但兩者間的誤差就很明顯了。而出現此現象的可能原因可能為 CPU 與其他程序的爭搶狀況導致拖累(TIME_MEASUREMENT 一次只有一個子程序可被執行，相較之下 FIFO_1 中尚為完成的子程序皆是處在被 fork 後再被 block 的狀態)，亦或是 lock 與 wake 導致 context_switch 所造成的誤差。

而各自內的誤差則可能是由正在等待的 CPU 數量、上述的同步問題、虛擬機環境等因素造成的，不過相較之下不是大問題。