OS Project 1 report

資工二 B07902003 吳雲行

執行方法

```
make
./main
```

設計

採取雙core的設計，scheduler會佔據一個core，執行process則用另一個。每單位時間後會評估是否需要生成process、換process跑或繼續執行正在執行的process。
函數分別用處：
main: 主要的scheduler，包含讀取input、將process按ready time排序、規劃next_process應該是哪一個，並呼叫其他函式進行schedule或execute等功能。其中執行四種policy:

1. FIFO - 為non-preemptive的排程，因此每次從ready的process裡面挑ready_time最早的做，一次將這個process做完才換下一個，因為一開始sort過所以基本上就是按照順序做完。
2. RR - 從ready的process裡面輪流做，當一個process做了超過500個單位時間時換成下一個process，若提早做完則從頭找下一個ready的process來execute。rr_time為前一次做輪轉的時候的時間，和ntime(目前為止進行的單位時間量)的差距可用來計算是否到了輪轉週期500。
3. SJF - 從ready的process看exec_time來排，為non-preemptive的排程所以只要選了就要一次做完。
4. PSJF - 和SJF排序方法類似，都是看exec_time，但SJF為non-preemptive的排程所以只要選了就要一次做完，PSJF則每次都要從ready的process檢查是否有exec_time較少的，則換成該process來執行。

setcpu: 規劃該process是使用哪一個CPU，schduler在CPU1，生出來的process則在CPU2。用sched_affinity實做。

wake: 將該process設成SCHED_OTHER，相較於下一個block函式中的SCHED_IDLE來說優先度較高，我們用這兩個的差別來實現只讓該process運行而其他的為block住

block: 將該process設成SCHED_IDLE來讓他被block而不被執行

execute: fork出一個小孩來跑該process所需的exec_time，搭配wake和block使用來讓他在輪到他的時候才執行。同時我們也使用到signal來讓他不要一被fork出來以後就偷跑，導致race condition。

另外解釋兩個system call的用處：
my_start(334): 用getnstimeofday來在一開始獲得開始時間，利用乘上1000000000的方式可以只以long形式回傳目前時間的秒數和奈秒數
my_end(335): 讀入開始時間和pid，並獲得結束時間後按照題目所需格式用printk印在dmesg裡。

核心版本

用uname -a得到:
Linux cloud-virtual-machine **4.14.25** #13 SMP Mon Apr 27 21:24:15 CST 2020 x86_64 x86_64 x86_64 GNU/Linux
linux版本則為**16.04**

實際結果和理論結果的差異與原因

實際結果會較理論結果還多一些時間，除了中間process在轉換時的多餘時間可能不算在內之外，整個VM也仍有其他process在運行，不免中途可能context switch到其他非此程式相關的process，即會影響結果。但基本上為小誤差，若排程正確的話不會影響我們最後輸出的排程結果

的順序。