Operating System Project1 Report

資工三 蘇庭葦 B06902130

環境&核心版本

OS : Ubuntu16.04
GCC version : 5.4.0
Linux-4.14.25

執行方法

make all 編譯所有檔案
sudo ./main 或 sudo make run 可以執行檔案
make clean 刪除編譯好的執行檔

設計

建立虛擬機時，分配了兩個CPU於此虛擬機。自行定義實作取得系統時間和印在dmesg中的syscall（getnstimeofday, printk），然後重新編譯kernel並測試syscall沒有問題。

首先main function從standard input將policy, process資訊讀入，確認policy無誤後將process資訊放入在標頭檔中定義好的process結構，然後呼叫runSchedule()進行排程。

一開始先把所有的process依照ready time由小而大用qsort進行排序。

為了不讓排程跟跑process搶同一CPU，所以將使用一個CPU專門排程，另一個CPU專門跑被排程的process。把runSchedule()自己的affinity先設定在0號CPU，然後為了確保1號CPU可以在控制中不會被安排去跑別的東西，要在1號CPU建立一個default runner的存在，使用fork()把這個process分支出去作為default runner，然後把default runner的affinity設定在1號CPU。

後面將透過調整priority然達到排程的作用，一次只會有一個process的priority明顯大於其他的process，而此process將會獲得使用1號CPU的優先權。

- priority調整方法：
- 優先執行者提高其priority，使用SCHED_FIFO(99)
- 等待執行者降低其priority，使用SCHED_OTHER(0)

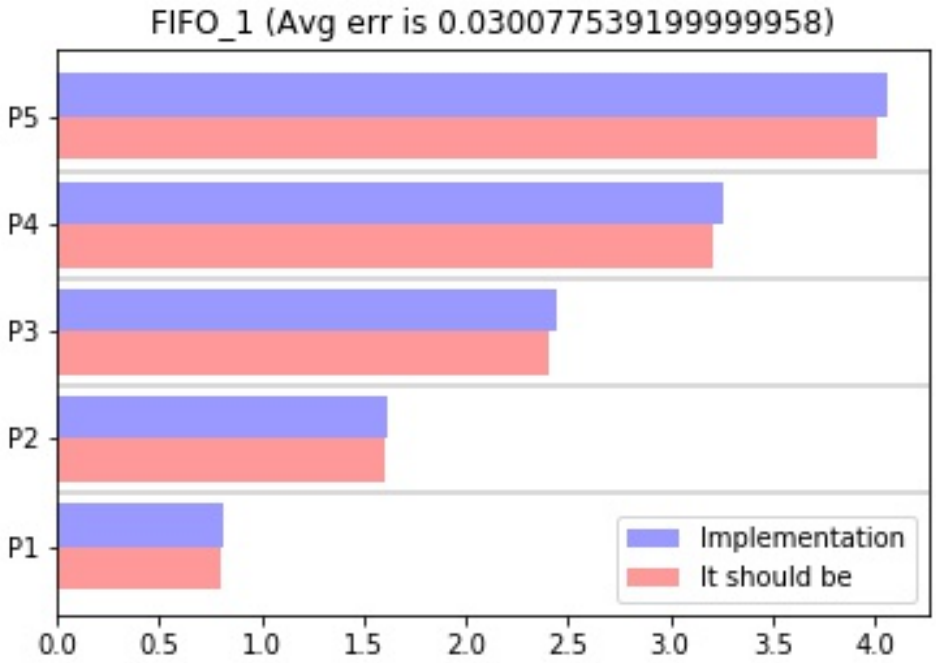
接著進入開始排程並執行process的while迴圈：

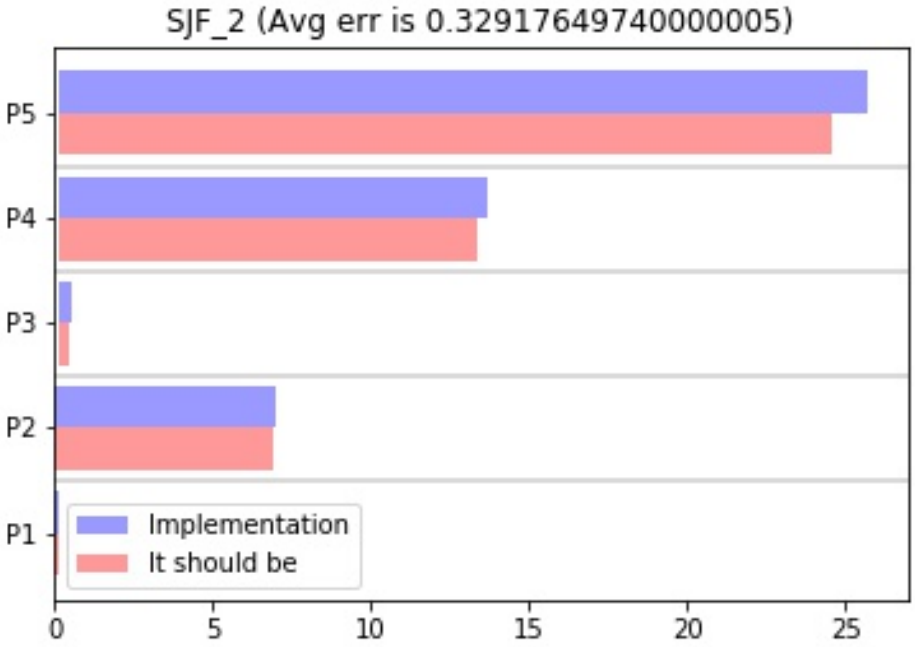
- 每一輪先判斷是否有已完成的process。
  - 有已完成的行程。wait()後提高default runner的優先度，紀錄已完成的行程數。
- 判斷是否有進到ready的process。
  - If a process is ready
  - fork()出子行程並將他的affinity設在1號CPU，然後push queue
  - fork()出的子行程執行如下
    - 用先定義的syscall get_time紀錄行程開始時間
    - 降低此行程的priority，待被提高後才開始執行後續
    - 執行對應的time unit
    - 用先定義的syscall get_time紀錄行程結束時間
    - 依照格式用先定義的syscall prints 將資訊印至dmesg
    - 用eixt()結束此行程
- 根據policy，判斷是否要打斷現在的process換另一個process開始跑。
  - FIFO：不會打斷正在跑的行程，先跑的就會跑完或主動放棄才會下一個進CPU。所以只要有行程正在跑，那這邊就可以return。如果是default runner在跑，才要把ready queue pop出來讓下一個進去跑。
  - SFJ：不會打斷正在跑的行程。但如果正在跑的是default runner，便從已ready的行程中選擇執行時間最短者成為下一個，提高其priority。
  - PSJF：檢查新加入的是否為剩餘執行時間最短者，否則即正在跑的process。如果是default runner在跑，如同SJF，便從已ready的行程中選擇執行時間最短者成為下一個，提高其priority。
  - RR：檢查此行程距開始時間是否已經過500 time unit。如果已達500 time unit，pop ready queue使其成為下一個行程，如果上一個行程還沒跑完再push進queue。
- while迴圈的每一輪及代表一個time unit，更新時間資訊
  - 執行time unit
  - 紀錄開始排程後經過的時間
  - 正在執行的process執行時間減一
- 所有process完成後，將default runner殺死，然後跳出while並結束。

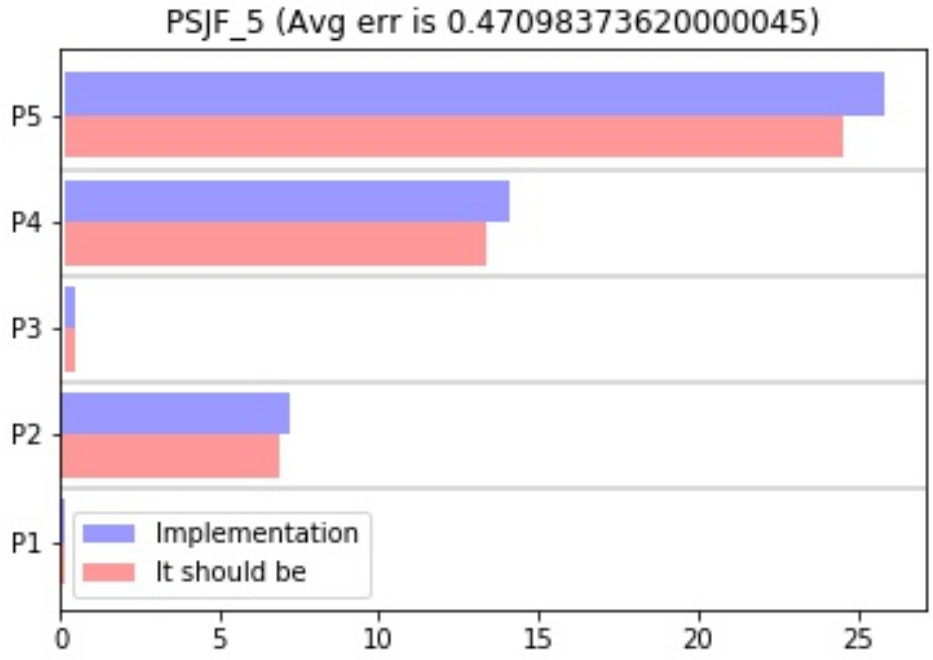
實際與理論的比較

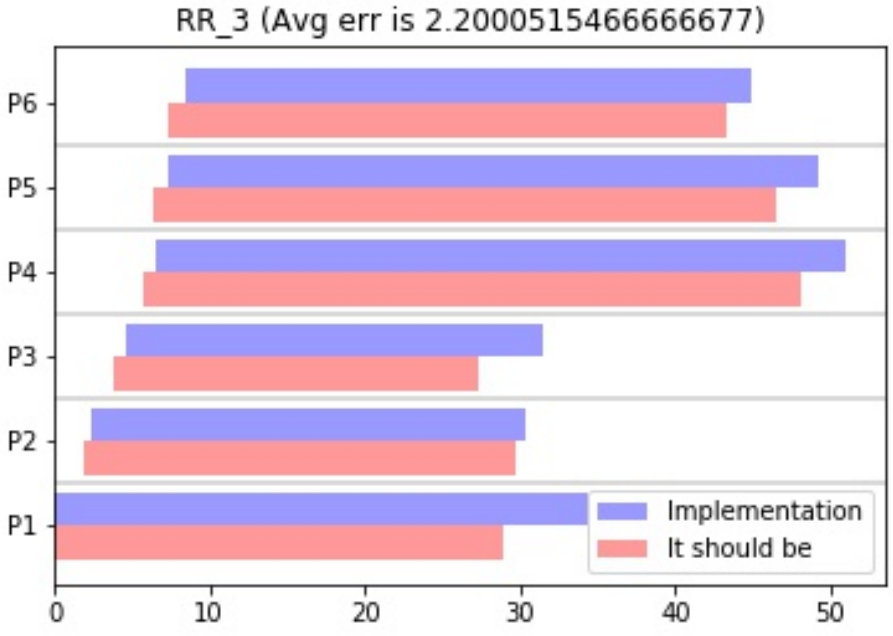
由TIME_MEASUREMENT.txt的結果，可計算得到平均一個time unit是0.78~0.9左右。
以time unit來說，變動範圍偏大。原因應該是電腦同時執行很多程式，呈現較忙碌的狀態。

以下以圖表方式呈現實際與理論的差別：
（開始時間是從fork()後開始計算）






PSJF_5 (Avg err is 0.47098373620000045)
P5
P4
P3
P2
P1
Implementation
It should be
0
5
10
15
20
25


RR_3 (Avg err is 2.2000515466666677)
P6
P5
P4
P3
P2
P1
Implementation
It should be
0
10
20
30
40
50

由圖表可以明顯發現實際所需的完成時間皆大於理論上的所需時間。
推測為以下原因：

- 實際上電腦還有跑其他程式，而且沒有其他CPU了。所以有時候會被切去跑其他的程式，並非一直在跑process，而以上的設計並沒有排除這一點。因此時間誤差受當時電腦忙碌程度影響。
- 以調整priority來做到排成作用，但當我的程式調整完priority後，真正的CPU不會馬上進行交換，會比預計的稍晚了一些，所以導致正在跑的process會多跑一些時間。
- 子行程經過的time unit和排程的行程經過的time unit有些誤差，所以如果執行時間越大，時間差異也會逐漸變大。

＊特別感謝 圖表協助製作者b06902110 林奕萱