OS Project 1

ID： b07902133 彭道耘

1. 設計

- System Call：我一共新增了三個 system call，前兩個用於 project 需要，而最後一個則用於測試：
  - my_get_time.c：
    - system 編號：333
    - 傳入一個 struct timespec 結構，並會藉由 getnstimeofday 函數，得到西元 1970 年至今的秒數，分別以兩個變數儲存整數部分以及小數部分。
  - my_print_info.c
    - system 編號：334
    - 一共傳入五個參數，分別為：
      1. pid： process id
      2. process 開始的秒數
      3. process 開始的奈秒
      4. process 結束的秒數
      5. process 結束的奈秒
    - 而接下來這個 system call 便會將這些資訊印到 kernel log 中。
  - my_hello.c：
    - system 編號：335
    - 測試用，會將 `Hello World!` 印到 kernel log。
- Scheduler：
  - scheduler.c / scheduler.h
    這是整個 Scheduler 的入口點，會負責處理從 stdin 讀取輸入，並將輸入依照 ready time 由小到大排序。接下來分類要執行的是哪一種排程方式，呼叫對應的函數執行。
  - process.c
    這個檔案一共包含了各種不同的函數以及排程方式，以下各自分述：
    - Process 管理：
      - `SetPidCPU(pid_t pid, int cpu_id)`
        指定要將 `pid` 這個的 process 執行在 `cpu_id` 這個 CPU 上面。
      - `SetPidPriority(pid_t pid, int priority)`
        指定要將 `pid` 這個的 process 的優先權設置為 `priority` 。
      - `StartRunProcess(struct Process *now)`
        收到 `now` 這個 process，接下來會 fork 出 child process 。 Fork 完成後會將 child process 的優先權調到最低，並等待這個 child process 開始執行。

- 各式排程：
  - `SchedulerFIFO(struct Process *ps, int n)`
    依照 FIFO 的方法來實現排程。每當一個 process 執行完畢，便會去找下一個 process 開始執行。當完成一共 n 個排程後，便會結束排程。
  - `SchedulerRR(struct Process *ps, int n)`
    使用了一個名為 `rrcnt` 的變數來記錄當前的 process 已經執行了多久，當 `rrcnt == 500` 或是當前 process 完成執行時，便會尋找下一個尚未完成的 process 執行。在這之中有使用一個 queue 來維護已經抵達 ready time 但尚未完成的 process 們。每當收到一個新的 process，便會將這個 process 加入到 queue 的尾端 (tail) 後面，並每次從 queue 的前端 (head) 取出 process 執行 500 個 unit time，接下來若還沒執行完則會將這個 process 的優先權調低並移到尾端 (tail) 後面。
    當完成一共 n 個排程後，便會結束排程。
  - `SchedulerSJF(struct Process *ps, int n)`
    實作時與 FIFO 的排程相當類似，不同的是當一個 process 被完成執行時， SJF 選擇的不是直接選下一個 process 執行，而且從已經 ready 的 process 中選擇需要執行最短的 process 來執行，在實作時維護了一個 status 的陣列紀錄每個 process 的狀態分別是已經完成、尚未 ready 或是已經 ready。當完成一共 n 個排程後，便會結束排程。
  - `SchedulerPSJF(struct Process *ps, int n)`
    實作時與 SJF 的排程相當類似，不同的是當一個 process 被正在被執行的當下倘若有一個 process 這時候完成了 ready，並且這個 process 有更短的執行時間，那麼這個時會會將原先正在執行的 process 的優先權調低，並讓新的 process 以高優先權先執行。每個一個 process 完成執行後便會從尚未完成的 process 中挑出執行時間最短的繼續執行，直到一共完成 n 個排程後，便會結束排程。

2.核心版本

- Kernel： Linux ubuntu 4.14.25
- Server： Ubuntu 16.04

3. 比較實際結果與理論結果，並解釋造成差異的原因

從執行結果來看，會發現到實際的結果有時會比理論結果更好，有時會更差。在執行多次同樣的資料後，會發現到結果的輸出並不穩定，有時會稍快或稍慢，推測是出自於以下幾個原因：

- Scheduler 再進行排程的時候本身也需要進行一些運算，包括排序等各種計算與維護演算法，在這之中都會花費一些時間，造成實際結果比理論結果還要來的久。
- TIME_MEASUREMENT 的估計誤差，由於是以 TIME_MEASUREMENT 的執行結果作為 UNIT_TIME 的標準，倘若在這之中高估了 UNIT_TIME ，便會造成誤以為實際結果比理論結果更快，而這個情況的發現主要是源自於 UNIT_TIME 的難估計造成的。
- 系統在執行期間同時也會運行其他的 process，且在不同時間裡面運行 process 也是不等量的，因此會對時間的估計造成一些誤差。