報告 化學四 廖哲宏 b05203008

1. 設計

程式分三部分：

- Main.c
- Scheduler.c
- Process.c

三者分別處理讀取資料、scheduling、process 處理。

- Pseudo code :

```
Scheduling(struct process *proc, int proc_num, int policy){
    Sort by ready time
    While(true){
        If (a process finished){
            Wait child to return
            Printf (process info)
            If (all process finished)
                terminate
        }
        Push ready process to ready state (first execute, then block it)
        Select next process to execute
        If (next process != current process)
            Context switch
        Time elapse( 1 unit )
    }
}
Next_proc(struct process *proc, int proc_num, int policy){
    If non-preemptive && not finished
        Continue
    If SJF || PSJF
        Find next shortest remaining execution time
    If FIFO
        Find the next earliest process
    If RR {
        if (a process finished || time quantum is up)
            Find the next ready process
        Else
            continue
    }
}
```

2. 核心版本

   x86_64, linux 4.14.25 via virtualbox

3. 比較實際結果與理論結果，並解釋造成差異的原因

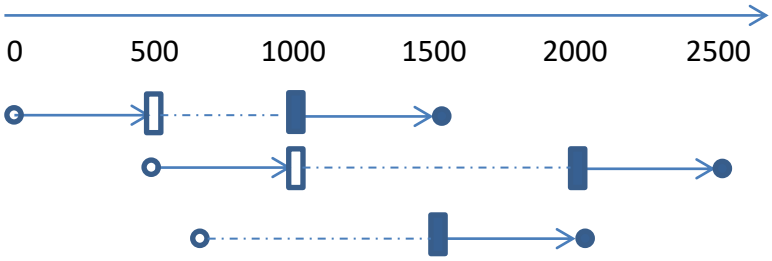
   在將 ready 的 process 執行的時候，我是將其 fork 出去執行後，讓他的 pid 存在原本已經預設好的 list 中(尚未開始執行的 process pid = -1)，但這樣在 RR 中可能會造成插隊的現象。舉例來說，若有一個 input 為：

   P1 0 1000

   P2 0 1000

   P3 600 500

   則甘特圖應為



   因為在 P2 開始執行前，P1 達到 time quantum 而排到 queue 的末端，這時 queue = {P2, P1}，而後 t = 600 時 P3 才進來，因此應該排在最後，即 queue={P2, P1, P3}。但我的作法是找尋下一個 ready 的 process，這會導致 P2 結束之後先找到 P3，而不是 P1。可能的解決辦法是利用 linked list 來存執行中的 process，並且將準備執行的 process 加入 linked list 的 tail。另一種方式為：將剛跑完的 process 的 ready time 重設為當下的時間，並且每次尋找 ready time 最早的 (最接近起始點)。這個方式直接利用 ready time 當作 priority queue，並且在 processru 結束一個 time quantum 之後保證退到 lowest priority，進而避免一開始提到的情況，也不需要另外拿記憶體建立 linked list，或許會比較方便一些。