

# NTU CSIE 2020 Project 1

R08922128

林子欽

---

## 設計

---

### 核心概念

- 實現四個排程策略，並利用 Linux 的 `sched_setscheduler` 調整優先權值，使 `processes` 在一個 `cpu` 上依排班策略執行，紀錄 `processes` 的開始與結束時間，再與理論開始與結束時間做比較
- 使用兩個 `cpu`，一個進行排班策略的計算，一個執行 `child processes`
- 排班策略的部分，使用一個 `while loop`，每次執行一個 `while loop` 是一個 `unit time`，`loop` 裡面偵測此時是否有 `process ready`，如果有就把它 `fork` 出來並丟到另一個 `cpu` 且 `priority` 降到最低
- `While loop` 接下來尋找現在該執行的 `process`，`non-preemptive` 的策略去看現在有無 `process` 正在執行，如果有的話讓其繼續，沒有的話去抓 `next process` 並將其 `priority` 設為最高，使其執行。`preemptive` 的策略則每次都要去看所有 `process`，找哪個 `priority` 最高，使其執行
- 接著 `while loop` 執行一次的 `unit time for loop`，計算現在時間
- `While loop` 最後計算各個 `process` 經過這輪後剩餘執行時間

### main.c

- `main.c` 進行讀檔，將排班策略與需執行的 `processes` 的 `data` 記錄在一個 `struct process`，內容包含 `name`, `start_time`, `exec_time`, `pid` 並依排班測策略 `call scheduler`

### process.c

- 提供控制 `process` 的一些 `function`
- `assign_to_cpu(pid_t pid, int cpu_id)` 可以限制 `process` 在特定 `cpu` 上執行
- `unit_forloop()` 即 `process` 執行一單位的動作
- `set_priority(pid_t pid, int priority)` 可以調整 `process` 的 `FIFO priority`
- `fork_process(Process_Data process)` `fork` 出新的 `process` 並立即將其

priority 設為 1，且 assign 到另一個 cpu

- *run\_process(pid\_t pid)* 將 process 的 priority 設為 99 (最高)
- *stop\_process(pid\_t pid)* 將 process 的 priority 設為 1 (最低)

#### scheduler.c

- *schedule(int strategy, Process\_Data \* process, int process\_num)* 被 main.c 呼叫的 function，從這裡再去執行選擇的排班方式
- *get\_next\_process(Process\_Data \* process, int strategy, int last\_index)* 依照不同策略取得下一個應該執行的 process
- *FIFO(Process\_Data \* process, int process\_num)* FIFO 排班策略實現
- *SJF(Process\_Data \* process, int process\_num)* SJF 排班策略實現
- *PSJF(Process\_Data \* process, int process\_num)* PSJF 排班策略實現
- *RR(Process\_Data \* process, int process\_num)* RR 排班策略實現

## 核心版本

---

- Platform: ubuntu 16.04
- Kernel version: 4.14.25

## 理論與實際之結果比較

---

- 經由測試 TIME\_MEASUREMENT.txt 取得 unit time 平均，拿第一個取得 cpu 執行的 process 的 start time 為開始時間，計算理論值，比較 demo 裡面的四個 case
- 基本上實際值不管在 start time 和 end time 都是比理論值大的，且越晚的 start time 和 end time 誤差將會越大，這是因為排程的 process 在 while loop 裡面需要做很多雜事，例如檢查有沒有新的 ready process、調整現有 process priority、找下一個該執行的 process 等等，所以一個 while loop 裡面並不是只有做一次 unit time for loop，做雜事時會浪費 child processes 所在的那個 cpu 的 cpu time，因此時間越久誤差將會越大

- 少數幾筆 process 的 start time 實際值小於理論值，這可能是因為每當我 fork 出 child process 時，在 parent process 將 child priority 調低之前，child process 因為繼承了 parent priority，因而搶先使用了一些 cpu time，start time 就開始計算了，所以會小於理論值

#### FIFO\_1.txt

理論：

P1: 1588057690.3539937 1588057691.435029

P2: 1588057691.435029 1588057692.5160644

P3: 1588057692.5160644 1588057693.5970998

P4: 1588057693.5970998 1588057694.6781352

P5: 1588057694.6781352 1588057695.7591705

實際：

P1: 1588057690.353993602 1588057691.454784905

P2: 1588057691.454900689 1588057692.554419819

P3: 1588057692.554536336 1588057693.648743656

P4: 1588057693.648860725 1588057694.749522729

P5: 1588057694.749635226 1588057695.842512027

#### PSJF\_2.txt

理論：

P2: 1588059077.5885088 1588059079.7505796

P1: 1588059075.4264383 1588059084.0747209

P4: 1588059086.2367914 1588059090.5609326

P5: 1588059090.5609326 1588059092.7230031

P3: 1588059084.0747209 1588059099.209215

實際：

P2: 1588059077.606446672 1588059079.804402663

P1: 1588059075.426438347 1588059084.155777703

P4: 1588059086.324901320 1588059090.694395871

P5: 1588059090.694502886 1588059092.886400074

P3: 1588059084.155872458 1588059099.452771696

RR\_3.txt

理論：

P3: 1588059778.655378 1588059808.9243667

P1: 1588059772.1691663 1588059811.0864372

P2: 1588059775.4122722 1588059813.248508

P6: 1588059787.3036606 1588059830.5450728

P5: 1588059785.1415899 1588059834.869214

P4: 1588059782.9795194 1588059837.0312846

實際：

P3: 1588059778.734024335 1588059809.123133166

P1: 1588059772.169166226 1588059813.479320959

P2: 1588059775.440711986 1588059814.572037923

P6: 1588059785.288243679 1588059830.857566409

P5: 1588059784.195907040 1588059835.208628756

P4: 1588059783.99053727 1588059837.438399007

SJF\_4.txt

理論：

P1: 1588058790.1042945 1588058796.5905063

P2: 1588058796.5905063 1588058798.752577

P3: 1588058798.752577 1588058807.4008594

P5: 1588058807.4008594 1588058809.56293

P4: 1588058809.56293 1588058813.8870711

實際：

P1: 1588058790.104294652 1588058796.661887735

P2: 1588058796.661987763 1588058798.862160676

P3: 1588058798.862265595 1588058807.733353807

P5: 1588058807.733458045 1588058809.949808201

P4: 1588058809.949918482 1588058814.372763970