OS-Project1

I. Scheduler design concept

- Overview

  程式主要分成兩個部分 scheduler 跟 process，分別模擬 scheduler 和 各個 process 的實作，另外為了讓 scheduler 能持續進行，並有效管理 process，因此透過以下的方法將 scheduler 跟各個 process 放在不同 core 上執行。

```
int set_task_to_core(int pid, int core);
```

- Scheduler

  Scheduler 統一管理 policy 跟 process，由於不論何種排程方式，process 都必須要是在 ready state 才可以執行，依序做了以下內容：

  1. qsort 將讀入的 process 照 ready time 排序
  2. 指派一個 core 給 scheduler
  3. 讓其他的 process 共用另一個 core 以確保 scheduler 能持續運行
  4. 根據 policy 是 RR FIFO SJF PSJF 中何者，來決定排程方式

```
int fifo_task(struct task *tasks, int task_all);
int sjf_task(struct task *tasks, int task_all);
int psjf_task(struct task *tasks, int task_all);
int rr_task(struct task *tasks, int task_all);
```

  5. 設計一個計時器來計算每個 process 運行的時間，以決定是否打斷

- Process

  不論是哪一種排程方式，都會有相同的 task/process 操作，包含:

```
int task_build(struct task task);
int set_low_priority(int pid);
int set_high_priority(int pid);
```

  因此在 task.c/task.h 的程式中也提供了這 API 提供 scheduler 控制各個 process，以確保能準確掌控每個 process 的狀態，並讓排程方式能遵照 policy。

II. Scheduler methods

- **First in first out (FIFO)**

  FIFO 選擇下一個 process 的方式，是根據每個 process 的 ready time，先 ready 的先 run。由於 FIFO 是 non-preemptive 的，如果今天有 process 正在 run，就等那個 process run 完。當沒有 process 在 run 的時候，可以跑一個 for 迴圈，判斷是否要輪到第 i 個 process run，判斷依據為，從還沒 run 完而且已經 ready 的 process 中，找 ready time 最小的，當作下一個要 run 的 process。

- **Shortest Job First (SJF)**

  SJF 選擇下一個 process 的方式，是根據每個 process 的 execution time，execution time 小的先 run。由於這個 SJF 是 non-preemptive 的，如果今天有 process 正在 run，就等那個 process run 完。當沒有 process 在 run 的時候，可以跑一個 for 迴圈，判斷是否要輪到第 i 個 process run，判斷依據為，從還沒 run 完而且已經 ready 的 process 中，找 exection time 最小的，當作下一個要 run 的 process。

- **Preemptive Shortest Job First (PSJF)**

  PSJF 排程的方法，和 SJF 方法很像，差別在於 PSJF 是 preemptive 的。每次直接從已經 ready 而且還沒 run 完的 process 中，找到剩下 execution time 最小的 process 來 run。當有 process 在 run 但 有另一個 process 剩下的 execution time 更小時，則該 process 先暫停，從還沒 run 完而且已經 ready 的 process 中，找出 execution time 最小的 process 來 run。

- **Round-Robin Scheduling (RR)**

  RR 排程的方法，和 FIFO 方法很像，差別在於 RR 是 preemptive 的，還有會定義一個 time slice，每一個 process 一次只能 run 一個 time slice 內的時間，當沒有 process 在 run 的時候，直接從已經 ready 而且還沒 run 完的 process 中，找第一個符合條件的 process 來 run。而當有 process 在 run 但 time slice 間隔時間到達時，則該 process 先暫停，從還沒 run 完而且已經 ready 的 process 中，找下一個 process 來 run。

III. Experiments & discussions

- FIFO_1.txt

```
P1 58
P2 59
P3 60
P4 61
P5 62
```

```
[project1] 58 1556594267.894986283 1556594268.866729461
[project1] 59 1556594267.895527934 1556594269.798723881
[project1] 60 1556594267.896376933 1556594270.731622774
[project1] 61 1556594267.903341439 1556594271.663572309
[project1] 62 1556594267.913586795 1556594272.536467136
```

理論時間 4.53s
實際時間 4.64148s

- FIFO_2.txt

```
P1 64
P2 65
P3 66
P4 67
```

```
[project1] 64 1556594272.597306849 1556594423.010700887
[project1] 65 1556594275.861007103 1556594431.912867663
[project1] 66 1556594276.060957870 1556594433.497014293
[project1] 67 1556594276.250956528 1556594434.709903664
```

理論時間 157.644s
實際時間 162.1126s

- SJF_1.txt

```
P2 150
P3 151
P4 152
P1 149
```

```
[project1] 150 1556594821.822900177 1556594825.612148730
[project1] 151 1556594825.103043155 1556594827.483588996
[project1] 152 1556594825.292906374 1556594834.942699451
[project1] 149 1556594821.822521166 1556594847.894800036
```

理論時間 25.368s
實際時間 26.0719s

- SJF_2.txt

```
P1 154
P3 156
P2 155
P4 157
P5 158
```

```
[project1] 154 1556594848.163118173 1556594848.363846585
[project1] 156 1556594848.380031612 1556594848.787335082
[project1] 155 1556594848.163834876 1556594856.241094418
[project1] 157 1556594848.380151911 1556594863.680263463
[project1] 158 1556594848.380207884 1556594876.550090105
```

理論時間 27.7236s
實際時間 28.17006s

- PSJF_1.txt

```
P4 93
P3 92
P2 91
P1 90
```

```
[project1] 93 1556594533.553956392 1556594539.166072608
[project1] 92 1556594531.684465390 1556594546.613488616
[project1] 91 1556594529.814415582 1556594557.689313026
[project1] 90 1556594527.948932169 1556594574.199189870
```

理論時間 45.3s
實際時間 46.25025s

- PSJF_2.txt

```
P2 96
P1 95
P4 98
P5 99
P3 97
```

```
[project1] 96 1556594576.258438039 1556594578.123495014
[project1] 95 1556594574.385252482 1556594581.820645835
[project1] 98 1556594583.710194088 1556594587.436960075
[project1] 99 1556594587.450174155 1556594589.312396118
[project1] 97 1556594581.089744628 1556594594.787671949
```

理論時間 19.932s
實際時間 20.40242s

- RR_1.txt

```
P1 117
P2 118
P3 119
P4 120
P5 121
```

```
[project1] 117 1556594656.147834001 1556594657.121242588
[project1] 118 1556594656.148316308 1556594658.058668808
[project1] 119 1556594656.149075770 1556594658.991598455
[project1] 120 1556594656.155446631 1556594659.904164800
[project1] 121 1556594656.166404917 1556594660.800382089
```

理論時間 4.53s
實際時間 4.65255s

- RR_2.txt

```
P1 123
P2 124
```

```
[project1] 123 1556594661.936053186 1556594675.893124998
[project1] 124 1556594662.944417236 1556594678.679361165
```

理論時間 16.308s
實際時間 16.74331s

- Discussion

比較實際時間與理論時間，我們發現所有 scheduling 方法的實際時間均大於理論時間。其可能原因有以下幾種：

為了減少切換 process 時所產生的延遲，我們使用雙核心，但是即使使用多核心，虛擬機系統本身還是會佔用到兩顆核心的效能，因此我們在各個 scheduling 的實際時間都比理論時間來得長。

除此之外在 round robin 或是以 priority queue 為基礎的 OS 系統中，可能會因為 busy waiting 使得 process 在等待時不斷重複檢查條件，造成資源的浪費並影響效能。

IV. Contribution

陳宣佑 D06944005 - PSJF/RR

林懷宇 R06922032 - Scheuler

陳熙 R07922151 - Kernel file

吳欣翰 D05944004- FIFO/SJF

盧承億 B03902108 - Process/Utility