Operating System - Project1

B06902023 林恩廷

- 設計

這次的架構設計大致如下：

- main.c:

  讀取 input，parse 之後將參數丟給 scheduler 執行主要內容。

- util.h / util.c:

  定義與實作 用來表示單位時間的迴圈 與 依照ready time排列用的比較函式。

- process.h:

  定義用來表示 process 的 structure。

- process.c:

  實作操作 process 的函數，包括將 process 綁定到特定 CPU、產生新 process、叫醒特定 process 與 暫停特定 process。

- schedule.h:

  定義 Policy。

- schedule.c:

  實作作業主要部分，包括不同 policy 對應到的 scheduling 方法，詳細內容下述。

- 實作架構：
  - 這次的做法是分在兩個 CPU 執行，CPU0 負責跑 scheduler，CPU1負責跑各個 process。
  - process 中比較特別的部分: scheduler 使用系統內建的 FIFO policy，並透過修改 priority 成 99/1來決定 process 應該 執行/暫停，但是過程中發現即使將 priority 設成 1 也有可能讓該 process 執行少量的運算，因此我在 process 的最開始加了 warmup 的迴圈(執行少量的單位迴圈)，避免 process 本該停止卻不慎執行到記錄start time的程式碼。
  - scheduler流程: scheduler 會執行一個無窮迴圈，每一次的迴圈就代表一個單位時間，在每個時間點首先會檢查目前執行的 process 是否結束，結束的話就將其回收並檢查測資有沒有全部執行完畢。接著檢查是

否有新的 process 已經 ready，如果有的話就先把它 fork 出來，並且讓他處於暫停狀態待之後排程時喚醒。接下來 scheduler 會決定下一次迴圈要執行的 process，根據不同的 policy 以及不同的時間點，可能會有保持原狀與進行 context switch 兩種可能，而 context switch 的實作就是讓執行中的 process 暫停，並喚醒接下來的 process。當每個 process 都被執行完之後，離開迴圈並且印出要求的 process name 與 PID 的對應。

- 核心版本

- Ubuntu 16.04
- Linux 4.14.25

- 比較

- 首先執行 TIME_MEASUREMENT.txt 來取的平均的單位時間，接著用 Gantt chart 手動計算每筆測資的理論解，最後觀察理論時間與實際時間之間的差異。
- 下面列出了部分測資的結果，可以看到實際執行時間會比理論時間來的長，最主要的原因之一是因為每次的迴圈內除了執行一個單位迴圈之外，還會有各種判斷式或是 context switch 等其他工作，若適當地減少這裡單位迴圈的次數(比如改成0.95個迴圈)可能可以減緩這個部分的問題(實際上並沒有這樣做，原因下述)。
- 其他造成誤差的原因主要有二：一個如同架構中所提到的，將 priority 設成1並沒有辦法真的完全使該 process 停止執行，因此會導致實際上仍然有少量的”偷跑”情形發生；另一個原因則是來自於 scheduler，因為每種策略，在不同的時間點所做的判斷的運算量不盡相同，因此每一次 scheduler 叫醒/暫停 process 時所產生的延遲都不太一樣，這也是我沒有直接調整每個迴圈中執行的單位時間的原因。

- 實驗部分（與Demo相同）

- TIME MEASUREMENT (可以看出單位時間大約為 1.3s / 500次)

```
P0 5223 [ 3235.573800] [Project1] 5223 1588053530.263334629 1588053531.588292410
P1 5224 [ 3238.080526] [Project1] 5224 1588053532.817183270 1588053534.096270205
P2 5225 [ 3240.752174] [Project1] 5225 1588053535.357886007 1588053536.769254225
P3 5226 [ 3243.344394] [Project1] 5226 1588053538.037198117 1588053539.362771757
P4 5227 [ 3245.917362] [Project1] 5227 1588053540.671755452 1588053541.937026182
P5 5228 [ 3248.568147] [Project1] 5228 1588053543.268308354 1588053544.589135919
P6 5229 [ 3251.188025] [Project1] 5229 1588053545.939556477 1588053547.210324431
P7 5230 [ 3253.799672] [Project1] 5230 1588053548.567036721 1588053549.823277665
P8 5231 [ 3256.543690] [Project1] 5231 1588053551.220133546 1588053552.568666509
P9 5234 [ 3259.516578] [Project1] 5234 1588053553.969340522 1588053555.543039517
```

- **FIFO_1**

```
P1 5358 [ 4985.673113] [Project1] 5358 1588055280.723607769 1588055282.139084930
P2 5359 [ 4987.162404] [Project1] 5359 1588055282.165763767 1588055283.628374015
P3 5360 [ 4988.570497] [Project1] 5360 1588055283.689207783 1588055285.036469211
P4 5361 [ 4989.986731] [Project1] 5361 1588055285.113778110 1588055286.452703358
P5 5362 [ 4991.432470] [Project1] 5362 1588055286.559602452 1588055287.898442108
```

時間比較	P1	P2	P3	P4	P5
理論/實際	1.3s / 1.4s	1.3s / 1.4s	1.3s / 1.3s	1.3s / 1.3s	1.3s / 1.3s

可以看出在基本的FIFO上表現得與理論值接近。

- **PSJF_2**

```
P1 5464 [ 6803.674750] [Project1] 5465 1588057096.312525131 1588057100.241947144
P2 5465 [ 6810.421301] [Project1] 5464 1588057090.776087319 1588057106.988498783
P3 5466 [ 6819.916406] [Project1] 5469 1588057109.971207448 1588057116.483604305
P4 5469 [ 6823.589918] [Project1] 5470 1588057116.484039296 1588057120.157114107
P5 5470 [ 6832.728529] [Project1] 5466 1588057106.988947468 1588057129.295725935
```

時間比較	P1	P2	P3	P4	P5
理論/ 實際	10.4s / 16.2s	2.6s / 3.9s	18.2s / 22.3s	5.2s / 6.5s	2.6s / 3.7s

可以看出在這個task中誤差比較大，但是整體順序正確且比值接近，大約在1.5倍的程度，只有P3、P4的比值比較小，因此可以推測這裡的誤差是源自於 context switch 以及 過程中相對大量的判斷式所造成。

- **RR_3**

```
P1 5508 [ 7687.654143] [Project1] 5510 1588057936.259922480 1588057984.221340446
P2 5509 [ 7690.679194] [Project1] 5508 1588057923.027603855 1588057987.246389948
P3 5510 [ 7693.120334] [Project1] 5509 1588057931.148222642 1588057989.687531545
P4 5511 [ 7735.283107] [Project1] 5513 1588057948.240635467 1588058031.850303729
P5 5512 [ 7744.573311] [Project1] 5512 1588057946.750078867 1588058041.140506692
P6 5513 [ 7748.759267] [Project1] 5511 1588057945.114831385 1588058045.326463427
```

時間比較	P1	P2	P3	P4	P5	P6
理論/ 實際	45.5s / 64.2s	42.9s / 58.5s	32.5s / 48s	61.1s / 100.2s	57.2s / 84.4s	50.7s / 75.1s

可以看出同樣因為頻繁做 context switch 的關係使得誤差較大，而理論時間與實際時間的比值同樣也大約落在1.5倍左右。

- **SJF_4**

```
P1 5577 [ 9074.469631] [Project1] 5577 1588059360.387939976 1588059371.138264251
P2 5578 [ 9077.305688] [Project1] 5578 1588059371.138599746 1588059373.974321431
P3 5579 [ 9091.146289] [Project1] 5579 1588059371.138490326 1588059387.814910902
P4 5581 [ 9093.969975] [Project1] 5582 1588059387.815302792 1588059390.638607845
P5 5582 [ 9099.617039] [Project1] 5581 1588059390.638822163 1588059396.285660251
```

時間比較	P1	P2	P3	P4	P5
理論/ 實際	7.8s / 10.8s	2.6s / 2.8s	10.4s / 13.9s	5.2s / 5.6s	2.6s / 2.8s

可以看出在這個task中，誤差較大的出現在理論時間較長的P1、P3，這是因為SJF是non-preemptive的，因此這些誤差主要來自於多次的迴圈中判斷式累積出來的延遲時間。