OS Project1

B06902069 資工三 許博翔

1. 編譯及執行

```
make
sudo ./scheduler < input_file
```

2. 設計

(1) scheduler.c

主程式。用來讀取 input，並且針對 policy 呼叫對應的 function (FIFO、SJF、PSJF 或 RR)。

(2) control.c/.h

定義 struct (記錄 process 的、手刻的 queue、手刻的 priority queue)、參數(自訂 syscall 的編號、一單位時間) 及常用 function(利用 sched_setscheduler 將 process 的 priority 設為 1 或 99 使之暫停或執行、更改使用的 core 確保 parent 在 core 0 及 children 在 core 1、手刻 queue 的 insert 及 pop...等)

(3) 自訂 syscalls

- 333 sys_my_gettime：給一個 timespec 的位址，將 getnstimeofday 的結果放在該位址。
- 334 sys_my_printk：給一個字串，printk 該字串。

(4) FIFO.c/.h

定義 scheduler 呼叫的 function。在 process ready 時將它放進 queue，當 CPU 閒置時取 queue pop 出來的 process 來執行直到結束。

(5) SJF.c/.h

定義 scheduler 呼叫的 function。在 process ready 時將它放進按照執行時間的 priority queue，當 CPU 閒置時取 priority queue pop 出來的 process (也就是現在以 ready 的 process 中執行時間最短的) 來執行直到結束。

(6) PSJF.c/.h

定義 scheduler 呼叫的 function。在 process ready 時將它放進 priority queue，這個 priority queue 用 heap 來維護，會把執行時間最短的放在最前面，當 CPU 閒置時取 priority queue pop 出來的 process (也就是現在以 ready 的 process 中執行時間最短的) 來執行，開始執行時初始一個 alarm clock，當 alarm clock 為 500 單位時間時，將執行中的 process 暫停，如果該 process 還需要執行，將它的執行時間扣掉 500 再放回 priority queue，然後取 priority queue pop 出來的 process 接著執行。若在 alarm clock = 500 之前結束，則直接取 priority queue pop 出來的 process 接著執行。

(7) RR.c/.h

定義 scheduler 呼叫的 function。在 process ready 時將它放進 queue，當 CPU 閒置時取 queue pop 出來的 process 來執行，開始執行時初始一個 alarm clock，當 alarm clock 為 500 單位時間時，將執行中的 process 暫停，如果該 process 還需要執行，將它放回 queue，然後取 queue pop 出來的 process 接著執行。若在 alarm clock = 500 之前結束，則直接取 queue pop 出來的 process 接著執行。

3. 核心版本

linux 4.14.25

4. 結果與比較

在 dmesg 中，記錄的是 process ready 的時間以及結束的時間。

經由 TIME_MEASUREMENT 得到的單位時間為 0.00193850560188 秒。

以下誤差以 |實際 - 理論| / 理論 來算。

(1) FIFO_1.txt

	實際	理論	誤差
P1	503.170	500	0.634%
P2	1042.489	1000	4.249%
P3	1550.469	1500	3.365%
P4	2067.002	2000	3.350%
P5	2601.477	2500	4.059%

(2) PSJF_2.txt

	實際	理論	誤差
P2	1094.612	1000	9.461%
P1	4318.920	4000	7.973%
P4	2134.565	2000	6.782%
P5	1074.393	1000	7.439%
P3	9585.168	9000	6.502%

(3) RR_3.txt

	實際	理論	誤差
P3	15842.246	14600	8.509%
P1	19488.710	18000	8.271%
P2	19336.189	17800	8.630%
P6	24142.133	22400	7.777%
P5	26910.511	25000	7.642%
P4	28502.841	26400	7.965%

(4) SJF_4.txt

	實際	理論	誤差
P1	3148.458	3000	4.949%
P2	3170.191	3000	5.673%
P3	6371.950	6000	6.199%
P5	2163.269	2000	8.163%
P4	6409.463	6000	6.824%

根據以上結果，可以看出實際執行時間較理論長，這可能是因為：

- 系統中其實還有其他 process，造成中間有些時間並非執行此作業裡的 process。
- 在 context switch 的時候，多少會延遲一段時間。
- parent 在進行 scheduling 的時候，也會花費一些時間，造成延遲。