2020 OS Project 1 — Process Scheduling

1. 設計

(1) 預處理：

將資料讀進來後，放進自己創建的struct process裡，而後按照ready_time來排序。最後依照指定的排程，呼叫不同function (ex. FIFO(), RR()…)

(2) function實作:

a. specify_cpu(): 為了避免父行程和子行程互相影響，故會用sched_setaffinity實做此函式，分別將父行程和子行程放到不同的CPU上。

b. block_proc(): 用sched_setscheduler(pid, SCHED_IDLE, &para)實作，降低process優先權。此函數設為inline function，加快調用速度。

c. wake_proc(): 用sched_setscheduler(pid, SCHED_OTHER, &para)實作，恢復優先權被降低的process。此函數設為inline function，加快調用速度。

d. child_proc(): 處理所有child process(被排程的process)該做的事，包含在最一開頭block自己、記錄開始時間、根據參數跑足夠的time_unit()、記錄結束時間，最後輸出答案到kernel 裡。

e. FIFO()、RR()、SJF()、PSJF():

- 根據當前時間fork已經ready的process
- 回收結束的process，並輸出訊息到stdout，若所有process都已結束，即結束程式
- 如果前一個process已結束或當前process可能被搶先(RR(), PSJF())，根據不同的排程方法，產生不同的條件，決定下一個process。執行wake_proc，並視情況block先前執行的process。
- 執行time_unit()
- 更新當前時間、exec_time(process剩餘執行時間)、round_time(for RR)等參數
- 重複執行上述步驟，直到程式結束

2. 版本

(1)核心： linux-4.14.25

(2)環境：Ubuntu 16.04

3. 實際與理論差異

我發現process們會有偷跑的現象：
根據我的設計， 在process到達ready_time，開始執行child process時，應該要先自己block自己，直到被wakeup才會記錄開始時間，接著執行跑unit_time的迴圈，但事實上，他們在被block住後，會往前偷跑一點點，然後才被block，因此記錄到的都是ready_time，而非在CPU上第一次被執行的時間。

```
[39920.985635] [project1] 8706 1588169913.792001115 1588170256.657979570
[39941.842408] [project1] 8707 1588169914.268627350 1588170277.514720131
[39945.572140] [project1] 8708 1588169914.728733404 1588170281.244477813
[39949.193130] [project1] 8709 1588169915.184707873 1588170284.865475730
```

另外，在跑FIFO_2.txt、P1執行它的80000個units時，也有其他process會夾雜在其中。只是這樣的現象並沒有嚴重到會影響執行結果。

```
8631 33100
8634 100
8631 33200
8631 33300
8631 33400
8631 33500
8631 33600
8633 100
8631 33700
8631 33800
8632 100
8631 33900
8631 34000
```

推測我block/wake process的方法可能有點缺陷，導致操控process的執行時會出現誤差，沒辦法徹底阻止不該有動作的process。