NTU CSIE OS 2020 Project1

- Student Name: 林楷恩
- Student ID: B07902075

a user-space scheduler based on the priority-driven scheduler built in Linux kernel for a set of child processes

Environment

- **Kernel Version**: 4.14.25
- **Platform**: Ubuntu 16.04

Execution & Tools

- 將kernel_files/裏面的檔案放到正確的地方，並重新編譯kernel
- 執行 `make`
- `sudo ./scheduler < OS_PJ1_Test/${test_name}`
- 一次跑完全部測試的 shell script: `run.sh` (需要 `sudo`)
- 比較理論與實際結果的 python3.6 script: `compare.py`

Design

Overall Structure

- 使用2顆 CPU，一顆(0)用來跑 Scheduler，另一顆(1)用來跑 children processes。
- scheduler.c 只負責 read and preprocess inputs, 將執行的 CPU 及 priority 調整好，就按照policy call對應的函數。對於每一種 Policy，其對應的 scheduler function 定義在獨立的檔案，例如 RR_scheduler 就定義在 RR.c 裏面。
- proprocessing 的部分包含將 processes 按照 ready time 升序排序，這樣能更方便後續處理 processes 的 arrival。如果出現 ready time 相等的情況，則按照在 input 裏的出現順序決定。這樣排序可以幫助 scheduler 更簡單有效率地決定在目前的時間點有哪些程式可以 run。

Process Control

我寫了一些 functions 來協助我進行 processes 的控制：

- `proc_set_priority`：利用 `sched_setscheduler` 設定指定的 priority。
- `proc_set_cpu`：利用 `sched_setaffinity` 讓指定的 process 跑在指定的 process 上

- `proc_start` : fork 出新的process出來，爲了避免他在 scheduler 還沒處理前先跑，一 fork 出來就立刻把它的 priority 設成最低。
- `proc_block` : 把一個 process 的 priority 調成最低。
- `proc_wakeup` : 把一個 process 的 priority 調成最高。
- `proc_term` : 把一個已經結束的 process wait 掉。

FIFO_scheduler

- 因爲在上面提到的預處理階段，已經把 processes 按照他們的 ready time 排序過了。所以我們可以直接依照他們在 array 裏的順序來處理。
- 維護一個整數 cur_p，負責處理「正在 run 的 process」。如果 cur_p 指到的 process 是 ready 的狀態，就把它跑起來。如果是在 running 的狀態，就把 remaining_time 減一。如果 remaining_time 爲0,就將其終止(wait)，並把 CPU assign 給下一個 process。

RR_scheduler

- 我用 Linked List 實作了一個 Queue，當一個 process 進入 ready state，就把他 push 進 queue 裏，如果當下沒有正在跑的 process，則 pop 一個 process 出來跑。
- 維護一個變數 `counter` 來記錄現在這個 process 還剩多少時間可以跑。每次把一個 process 叫醒，就把 `counter` 設爲 time quantum(500)。並在每個 time unit 之後減一。如果減到零了，且那個 process 的 remaining time > 0， 就把他 push 回 queue 裏，換下一個 process 跑。

SJF_scheduler

- 如果當下沒有在跑的 process，就從所有在 “ready state” 的 processes 中找出 remaining time 最低的 process 來跑。
- 每個 time unit 減少一次現在在跑的 process 的 remaining time，直到它變爲 0，終止程式並 release CPU。

PSJF_scheduler

- 如果當下沒有在跑的 process，就從所有在 “ready state” 的 processes 中找出 remaining time 最低的 process 來跑。
- 如果當下有在跑的 process 且它的 remaining time 大於在目前已經 ready 的 processes 中的最小 remaining time，就 preempt 現在這個 process，改成跑 remaining time 最小的那個。
- 每個 time unit 減少一次現在在跑的 process 的 remaining time，直到它變爲 0，終止程式並 release CPU。

比較實際結果與理論結果

- 基本上，實際上的 finish time 和理論上的 finish time 沒有差很多（小於5%），這代表我的 scheduler 有確實遵循對應的 policy。但在大多數的 case，實際結束時間比理論結束時間略短，我認爲這是因爲我的 scheduler 沒辦法完全控制每個 process 要不要跑，而造成有些 process 能

在 scheduler 不知道的情況下稍微「偷跑」一點。例如以下情況：process A 結束，但此時因爲計時誤差，scheduler 認爲 A 還沒跑完，所以沒有及時 assign 下一個 process。那麼在 A 結束到 scheduler assign 下一個 process 的這段時間，其他 process 就可能有機會佔用 CPU (因爲此時沒有 priority 比他們高的 process)。

- 有極少數情況，實際的 start time 會「遠早於」理論的 start time，我認爲這也是因爲我的 scheduler 無法在已經把一個 process fork 出來的情況下，仍能保證它不會在 scheduler 喚醒它之前佔用「少許」的 CPU。因爲獲取時間的 system call 所需的執行的時間很短，所以這少許的佔用，就足以讓它獲得一個很早的 start time，造成大量的誤差。對於這個問題，我有2個可能的解法：

    i. 這是我在最終採用的解法。利用 nice 會繼承的特性，在 fork 前把 scheduler 的 nice 設得很高（優先度低），而因爲我已經把 scheduler 設爲 real-time process，所以不會受到高 nice 值的影響。 所以最終只有 child process 會被 block 住。然而，實際上仍有少數情況仍會使 child process 佔用極短時間的 CPU，而出現實際的 start time 會遠早於理論的 start time的現象。

    ii. 利用一個 priority 比所有在 ready queue 中的 processes 還要高的「假程式」，去擋住那些不應該執行的 process。然而，我的實驗結果是，因爲這個假程式佔用過多 CPU，使得結果誤差過大。