Scheduler design

1. Data structure: 採用 linux kernel 的 doubly linklist 作為 ready queue 的 data structure
2. scheduler 和 processes 分別在不同的 cpu 上運作，以避免被 child process 的時間干擾
3. Timing:

   (system call) 分別紀錄開始時間、結束時間，並在 syscall 結束時 printk

   main process 有個 global timer 進行記錄當前時間
4. Policy:

   -FIFO: 將 process 依照 ready time 進行排序，再依照 ready time fork processes，並等待所有 processes 都完成。

   -RR: 將 process 依照 ready time 進行排序，並將 processes 的 policy 設為 SCHED_RR，再依照 ready time fork processes，實行 RR 策略，並等待所有 processes 都完成。

   -SJF: 將 process 依照 ready time 排序，當 ready time 相同，則執行時間較短的優先權較高，再依照 ready time fork processes，並等待所有 processes 都完成。

   -PSJF: 將 process 依照 ready time 排序，當 ready time 相同，則執行時間較短的優先權較高，再依照 ready time fork processes，當有 process ready 時檢查是否可以插隊(比較當前的執行時間)，或者到了 process 的結束時間，將 ready queue 的第一個 process 移除，並將剩餘 processes 中執行時間最短的放到 ready queue 的第一個插隊，以此類推，直到所有 processes 都完成。當 child processes 被 fork，會先檢查 ready queue，若 queue 為空，則代表沒有 process 在等待，故將自己的 process 設為 SCHED_FIFO；若不為空則檢查是否可以插隊，可以插隊的話將自己的 process 設為 SCHED_FIFO，若不能則設為 SCHED_IDLE 等待

Discussion

1. 使用 single core 來執行 child processes，避免多個 cores 影響執行順序，用主程式的 timer 來作為 fork processes 的依據，並計算出結束時間
2. 對時間的影響：

   - 若 fork 沒有馬上進入 OS 的 SCHED_FIFO，則會造成 delay，影響後面的 process

   - 若結束時間預測稍有延遲，導致延後 fork 下一個 process，就會造成 ready queue 中沒有 process，導致時間增加。(提早結束則無影響，因為只是提早進入 FIFO queue 中等待)