B07705054 簡博軒

1.設計

(程序紀錄的 start time 用程序產生(ready)的時間紀錄)

所有程序分成三種狀態：
waiting：尚未到 ready time，程序尚未生成。
ready：程序已生成，可能正在運行或是等待 scheduler 排程。
finished：已經做完。

使用兩個 CPU 模擬，其中一顆用來跑排程(scheduleCPU)，另一顆跑輸入的程序(processCPU)。
其中藉由 sched_setscheduler 調整 processCPU 上程序的 priority ，來決定要優先執行的程序，這部份使用 sched_setaffinity() 調整。

調整 priority 方法(透過 sched_setscheduler())：
提高：將欲提高程序設成 SCHED_FIFO
降低：將欲提高程序設成 SCHED_IDLE

排程部份 pseudo code：

```
1.將所有程序以 ready time 從小到大排序 若相同則照原本輸入順序
while (還有程序沒做完) {
    檢查當下時間是否有新程序生成。
        if (有新程序產生) {
            將該程序狀態改為 ready 、加入 readyQueue 並 fork 出子程序。
            降低子程序的優先順序。
        }
    if (SJF)
        若當前無程序在跑，選 readyQueue 中執行時間最短者。
    else if (PSJF) {
        若當前無程序在跑或有新程序加入 readyQueue，選 readyQueue 中執行時間最短者，
        必要時採取 context switch。
    }
    else {
        if (無程序在跑)
            if (readyQueue 不為空)
                選出 readyQueue 中最前面的。
    }
    若有選出新的程序運行，增加它優先順序。

    跑一單位時間。

    if (當前程序執行完畢) {
        waitpid;
        將程序狀態設為 finished。
    }
    else if (RR) {
        檢測是否超過 timequantum。
        若有，降低它 priority 並選出新程序。
    }
}
```

2.核心版本

3.實際與理論結果差異

以 FIFO_1 的結果為例，完整跑完這個測資總共花 4.9437 秒，實際上只花了約 2060 的執行時間(理論為 2500)，可能原因如下：

- 從程序生成時到它被指派到另一個 CPU 且降低優先順序，這段時間該程序可能或多或少有在 CPU 上運行，而造成實際數值減少。(此項因素我認為為影響誤差最大的要件，且不可預期，也與當時 CPU 正運行程序有關)

其他可能誤差：

- 每一次生成新程序時使用 fork 來達成，而每一次 fork 約需要 $\frac{1}{20}$ 的單位時間。
- 每次當執行完一個子程序，呼叫 waitpid 時，兩邊的計時器是不同步的，若 parent process 的計時器比 child process 早到達結束時間，則必須花額外的時間等待兩者同步，可能造成後面等待排程的程序時間上受到影響。