Operating Systems - Project 1

資工四 B05902023 李澤諺

1. 設計

(1) 我使用了 2 個 CPU，一個用來執行 scheduler，另一個則用來執行 child process，以免這兩者彼此互相干擾，此外，我使用 sched_setscheduler 將 scheduler 在電腦上的 scheduling policy 設爲 SCHED_OTHER。

(2) Scheduler 會執行一個 while(1) 迴圈，在每個迴圈中，scheduler 會先判斷現在是否有 child process 準備執行完畢，若有，scheduler 便會使用 waitpid 等待該 child process 完全執行完畢，以免產生 zombie process，接著，若所有的 child process 都已經執行完畢，scheduler 便會結束迴圈並結束程式。

(3) 在判斷完是否有 child process 執行完畢之後，scheduler 接著會判斷是否有 child process 已經 ready，若有，scheduler 便會 fork 出該 child process，並使用 sched_setscheduler 將該 child process 在電腦上的 scheduling policy 設爲 SCHED_IDLE，等同於將該 child process 暫停，在這之後，會將該 child process 加入 ready queue 中。

(4) Child process 被 fork 出來之後，會依據其 execution time 執行對應數量的 unit time，過程中，其應該要暫停或是繼續執行完全是由 scheduler 透過 sched_setscheduler 來控制，若 scheduler 將該 child process 在電腦上的 scheduling policy 設爲 SCHED_IDLE，等同於將該 child process 暫停，若 scheduler 將該 child process 在電腦上的 scheduling policy 設爲 SCHED_OTHER，等同於將讓該 child process 繼續執行，此外，在child process 已經 ready 並被 fork 出來以及其 execution time 結束的這兩個時間點，會使用我自己寫的 system call，去使用 getnstimeofday 取得時間，並在 child process 執行結束時使用我自己寫的另外一個 system call，輸出訊息到 kernel。

(5) Scheduler 在判斷完是否有 child process 已經 ready 之後，會根據我們指定的 scheduling policy 去尋找下一個要被執行的 child process，若有要被執行的 child process，便會將目前正在執行的 child process 暫停，並執行下一個要被執行的 child process，並更新 child process 目前執行了多少個 unit time。

(6) Scheduler 在做完以上的判斷之後，便會執行 for (volatile unsigned long int i = 0 ; i < 1000000UL ; i++)，經過一個 unit time 的時間，再進入下一個迴圈。

(7) 各個 scheduling policy (FIFO、RR、SJF、PSJF) 的原理如課程所述，不過在此仍講述一下我實作的方式

FIFO：在每個 unit time 中，scheduler 會先判斷目前是否有 child process 正在執行，若有，則讓該 child process 在下一個 unit time 中繼續執行，而若目前沒有 child process 正在執行，便從所有已經 ready 且還沒有被執行過的 child process 之中，選擇一個 ready time 最小的 child process 在下一個 unit time 中來執行，若 ready time 最小的 child process 不只一個，則選擇在 input 中最先出現的 child process。

RR：在每個 unit time 中，scheduler 會先判斷目前是否有 child process 正在執行，若有，且該 child process 目前已連續執行的時間還沒到達一個 time quantum，便讓該 child process 在下一個 unit time 中繼續執行，而若目前有 child process 正在執行但其已經連續執行了一個 time quantum 的時間，便將其加入 ready queue 之後，並選擇 ready queue 之中的第一個 child process 讓其在下一個 unit time 中繼續執行，進行 context witch，而若目前沒有 child process 正在執行，scheduler 也會選擇 ready queue 之中的第一個 child process 讓其在下一個 unit time 中繼續執行。若目前有 child process 已經 ready，且同時也有 child process 被暫停，皆要被加入 ready queue 時，則會先按照 input 的順序，將已經 ready 的 child process 依序加入 ready queue，接著再將被暫停的 child process 加入 ready queue。

SJF：在每個 unit time 中，scheduler 會先判斷目前是否有 child process 正在執行，若有，則讓該 child process 在下一個 unit time 中繼續執行，而若目前沒有 child process 正在執行，便從所有已經 ready 且還沒有被執行完畢的 child process 之中，選擇一個剩餘的 execution time 最小的 child process 在下一個 unit time 中來執行，若剩餘的 execution time 最小的 child process 不只一個，則選擇在 input 中最先出現的 child process。

PSJF：在每個 unit time 中，scheduler 會從所有已經 ready 且還沒有被執行完畢的 child process 之中，選擇一個剩餘的 execution time 最小的 child process 在下一個 unit time 中來執行，若剩餘的 execution time 最小的 child process 不只一個，則選擇在 input 中最先出現的 child process。若目前正在執行的 child process 和被選擇到的 child process 不同，便會將目前正在執行的 child process 暫停並加入 ready queue 之後，進行 context switch。若目前有 child process 已經 ready，且同時也有 child process 被暫停，皆要被加入 ready queue 時，則會先按照 input 的順序，將已經 ready 的 child process 依序加入 ready queue，接著再將被暫停的 child process 加入 ready queue。

2. 核心版本

Linux 4.14.25

3. 比較實際結果與理論結果，並解釋造成差異的原因

我發現 child process 實際上的執行時間比理論上的多出了許多，我認爲其原因爲 scheduler 和 child process 還要進行其它的邏輯判斷和運算，甚至當其使用 function call 或 system call 時，更會讓實際上的執行時間變得更久所致。