OS-Project 1: CPU Scheduling

- **設計：**

  a. 實驗方法/核心版本

  本次實驗使用 Linux Kernel 4.19.37 搭配 Ubuntu 16.04 OS System。利用修改（增加 system call)過的 Ubuntu 16.04 System call 取得系統時間並配合 Linux Kernel 4.19.37 原生文件來達到模擬計算各 Process Executiom Time。

  為了精準計算時間，利用 sched_setaffinity() 將 parent process 與 child process 分別 assign 至不同 cpu，以避免當 parent process 在計算 time unit 時 cpu time 被 child process 搶走。

  b. 增加系統 System call

  在 /home/linux-4.19.37/kernel 中實欲新增的 System call，即 SYSCALL_DEFINE3(hello_3,int,state,int,pid,timespec*,start) 並於/home/linux-4.19.37/arch/x86/entry/syscall_64.tbl 中增加 System call entry。

  此次 Project 增加取得系統時間的 system call，並在每次 fork() child process 時使用 syscall(336, 1, mypid, start) 取得開始執行時間，並於 child process 結束時利用 syscall(336, 0, mypid, start)取得結束時間。

- **FIFO / RR**

  將每個要執行的 child process 依照 ready time 進行 non-decreasing 排序， 並從 ready queue 中依 first in first out(ready time 最早的先完成)的規則抓取執行，同時 parent process 會以 Unit time 的標準計算時間來判斷 child process 執行了多少 time unit。

  RR(Round Robin)步驟同上,只是每個 child process 最長執行時間 (time quantum) 為固定的 500 time units。

- **SJF**

  將每個要執行的 child process 依照 ready time 與 execution time 進行 non-decreasing 排序, 排愈前面 priority 愈高. 步驟同 FIFO，差別只在於 ready queue 的排序方法(執行時間最短的先完成)。

- **Preemptive-SJF**

  步驟同 SJF，差別在於 Preemptive 版本會在規定的時間檢查 ready queue 中 child process 剩餘執行時間，並把擁有最短剩餘時間的 child process 插隊置於 ready queue 最前端。

- **比較實際結果與理論結果**

time Unit = 1unit/0.0013571963198s (小數點以後四捨五入)	**FIFO**	**PSJF**	**SJF**	**RR**
1	實際：2434 理論：2500	實際：22270 理論：25000	實際：13602 理論：14000	實際：2432 理論：2500
2	實際：85136 理論：87000	實際：9704 理論：11000	實際：14860 理論：15300	實際：8230 理論：9000
3	實際：22349 理論：23000	實際：2916 理論：3500	實際：31176 理論：32020	實際：26826 理論：30000
4	實際：2939 理論：3200	實際：4561 理論：14000	實際：10002 理論：11000	實際：21275 理論：23000
5	實際：22142 理論：23000	實際：14493 理論：15300	實際：3243 理論：3500	實際：21756 理論：23000

會造成實際值比理論值還小，猜測是因為 Time unit 的計算過大，導致實際值偏小。