2020 OS Project1 Report

姓名：徐豪
學號：d06944016

1. 設計：
   這次設計排程器使用不同的policy來做出不同的排程方式。在起始設定，父行程會單獨分配到cpu，然後在對其使用SCHED_OTHER，確保精準使用時間單位計算，且避免子行程出來後的時間干擾，而子行程會使用到另一個cpu core 確保可以讓排程實現真正的行為。

   此設計讓父行程和子行程分開運作之下，使用不一意的policy，因此子行程會單獨出現在同一個 ready queue 上，且不會交互使用在同一顆cpu core 的時間，所以父子行程有分開獨立且同時處理的效果，因此在跑計算時間的空迴圈，不會因為父子行程同時跑導致時間衝突到。

   大致排程方式如下：
   - FIFO：有特別設定不可被搶先，就是running 還沒跑到 -1 時，仍然執行在running所對應到的process index 直到執行時間歸零，在waitpid()結束呼叫且running設定成 -1，就會進行下一個行程進入cpu執行處理

   - SJF和PSJF：已經在ready time 時間後要處理的行程，會根據execute time 較短的優先處理執行，而SJF必須等到前一個執行完才能換下一個，所以行程不可搶先。 相反，PSJF可以執行行程搶先在cpu執行

   - RR：設定在500單位時間倍數的執行時間，強迫下一個行程可以進行搶先執行，而過500單位時間執行後還沒完成所有執行時間，此行程就會被排到後面，等待下一次執行。注意在每個行程限定只能執行500單位時間，就會切換下一行程，直到所有行程跑完為止

2. 核心版本：
   程式跑在acer的筆電主作業系統，而非在虛擬機器上執行，使用 ubuntu 16.04， 內核啟動版本為 4.14.25 linux kernel

   在此程式有先跑呼叫計時器（sys_execution_time.c）的system call，使用 getnstimeofday() 去計算秒數小數點以前及計算小數點後9位的精確秒數，即為奈秒

   做完kernel compile 的 實際步驟如下：
   - Copy kernel_file/sys_execution_time.c 到 ~/linux4.14.25/kernel/sys_execution_time.c
   - Add line "obj-y += sys_execution_time.o" in linux4.14.25/kernel/Makefile
   - Add line in "linux4.14.25/include/linux/syscalls.h": asmlinkage int sys_execution_time(int isExecute, int *pid, unsigned long *start_sec, unsigned long *start_nsec, unsigned long *end_sec, unsigned long *end_nsec);
   - Add “ 334 common execution_time sys_execution_time in linux4.14.25/arch/x86/entry/syscalls/syscall_64.tbl
   - sudo make -j4 bzImage
   - sudo make install
   - reboot
   - 在主程式目錄下 sudo make
   - sudo ./sched.out < Input.txt > Output.txt
   - dmesg | grep Project1

3. 做出的結果：
   其中21個test set放在output的資料夾裡，有stdout 跟 dmesg 的結果，測試stdout輸出txt檔有時會有輸出 process pid 重複輸出的結果 bug，但是在terminal上卻只會輸出一次，但最後看整體完成行程的順序跟dmesg的順序是吻合的，指的是看第一次輸出的process不考慮它重複印出的proess順序會與dmesg符合

   關於在測試結果誤差，會受ready time開始時間影響及執行多長(execte time)時間造成排程進入cpu的時間順序，因此，整體完成順

序不一定要照後面的process排程的理論會搶先的，但ready time還沒到所以就還沒進入ready queue所以無法搶先，如果都以在ready queue情況下，最短排程很明顯可以看到 execute time大的process會排在最後執行處理完成