Operating System Project1 Report

系級：電機四　學號：B05202061　姓名：陳威旭

1. 設計：
   A. Kernel file : 總共有兩個新的 system call 333, 334
      I. get time(333)：使用 getnsitimeofday，將 kernel time 回傳給 user 做紀錄。
      II. print dmesg(334)：傳入 start time, end time，並用 printk 把資訊印出來。

   B. Process block : 主要管理 process 用
      I. struct Process：存有 process 的 ready time, remain execute time, pid，且把讀進來的 input process data，分別放入各個 process block 內。
      II. assign core：將 scheduler process 放到 CPU2，child process 放到 CPU3。
      III. process kick：利用 sched_setscheduler，將 process 的 priority 設定最低。
      IV. process highest：利用 sched_setscheduler，將 process 的 priority 設定最高。
      V. process create：傳入 process execute time 並創造出 child process，使其直接開始一直 run time unit。

   C. scheduler：四大 method
      I. FIFO：由於助教說測資 start time 確定已排好，故我就直接依照順序，create process -> run -> waitpid，直到全部跑完。
      II. SJF：每次從已 start 的 process 中，選出 remain execute time 最短的，給他最高 priority 去跑，直到跑完，再去選下一個，或 stall 住，等到下一個可選 process 出現。
      III. PSJF：每當有任一個新 process 的 start time 到了，就暫停動作去判斷目前全部 process 中哪個 remain execute time 最短，讓他開始(繼續)跑，一直跑直到他跑完，或者被另一新 ready process 中斷。
      IV. RR：用一個 for 迴圈，一直依序確認 process 1~N，看誰想要跑，若 i 想跑，就讓他跑 500 Time unit，或跑完了提早結束，時間到了或結束後，就繼續確認 i+1 是否要做。一直輪流確認，輪流做，直到全部 process 跑完。
      V. 以上方式中都有可能出現某些時候沒 process 跑的狀況，或要好好判斷該次 process 要跑多久的狀況，故此時我運用了我通過 HW2

online judge 的 scheduler 經驗，來做了一些處理，快速找出下一個可 run 的時間點。

D. 輔助：read, write
   I. 在 RR 時，由於我怕 scheduler 和 child 的時脈會不同，故新增了此溝通機制。若無溝通機制，可能 schedule 跑了 500 個 time unit 後，child 才跑 505 個 time unit 而已，此時 scheduler 才去指定改變 priority，累積下來會使 child 需要較少次的 RR 輪到機會，就能真正跑完，雖然我都把正在跑的 process 的優先度設到最高(SCHED_FIFO)了，不過我是在虛擬機上跑，故我無法確定外面真正的電腦的 CPU 是否真的完全給虛擬機用了，故新增此溝通機制可以保險一些。
   II. 但就算拿掉此溝通機制(大概改動個 10 行左右即可)，整體結果仍會是正確的，我皆有進行測試過，只是沒有溝通機制時，若兩個 process 在理論執行完 time 非常接近時，可能會出現誰先真的執行完不太一定，理由同第一點所說。

E. 我的整體架構是有參考網路上學長的(老師有於 4/29 的線上討論說參考架構是 OK 的)，但我整個 code 是有重新打過的，只有架構一樣，內容都不一樣，且我運用了我 HW2 online judge 的排程經驗(似乎到目前還是只有我通過 OAO)，來減少 scheduler 的運算，使兩邊若不能溝通時，時脈可以盡量一致，加強 system call 的實用性，以及賦予 priority 時，給予要執行的 process 最高優先度(SCHED_FIFO：real time 的優先度)。

2. 版本：

- Using Linux kernel version 4.14.25
- Oracle VM VirtualBox version 6.1
- gcc version 5.4.0

3. 比較實際結果與理論結果(方便起見，各舉前三個當例子)

I. 先計算出實際 TIME UNIT

TIME_MEASUREMENT					
FIFO					
process	理論start	理論end	dmesg start	dmesg end	delta
P1	0	500	131748.322576529	131749.144013933	0.821437403996242
P2	1000	1500	131750.001001720	131750.918960762	0.917959042009898
P3	2000	2500	131751.734563402	131752.660747320	0.926183918025344
P4	3000	3500	131753.503730299	131754.450744952	0.947014652978396
P5	4000	4500	131755.301558876	131756.216231729	0.914672852988588
P6	5000	5500	131757.065592022	131757.978788576	0.913196554000024
P7	6000	6500	131758.830656042	131759.703122578	0.872466536005958
P8	7000	7500	131760.581960054	131761.477425588	0.895465534005780
P9	8000	8500	131762.305231129	131763.156941348	0.851710218994413
P10	9000	9500	131764.007813006	131764.897522618	0.889709612005390

A. 從 delta 可以得到 500 time unit 平均要跑 0.894981633 秒。(以此為準)

B. 但從整體 0 到 9500 的話，可以得到 500 time unit 平均要跑 0.872365584 秒。

C. 雖然我已經給予 process 有最高 priority 去使用 CPU 了，不過可能是因為在虛擬機上，外部真正的電腦，並沒有完全給予 CPU3 的使用時間，所以使 delta 的值，有點浮動，約在+-8%之間。

II. FIFO

FIFO_1							
process	理論start	理論end	dmesg start	dmesg end	delta	actually run unit	error(%)
P1	0	500	131960.622893836	131961.498554060	0.875660224002786	489.2056955	-2.159
P2	500	1000	131961.498819143	131962.357390655	0.858571512013441	479.6587331	-4.068
P3	1000	1500	131962.357640810	131963.301869139	0.944228328997269	527.5126856	5.503
P4	1500	2000	131963.302123602	131964.165110677	0.862987074971898	482.1255787	-3.575
P5	2000	2500	131964.165316201	131964.989036769	0.823720568005228	460.1885324	-7.962

FIFO_2							
process	理論start	理論end	dmesg start	dmesg end	delta	actually run unit	error(%)
P1	0	80000	132023.389673939	132164.074254287	140.684580347996000	78596.35061	-1.755
P2	80000	85000	132164.074514499	132173.280774487	9.206259988015510	5143.267556	2.865
P3	85000	86000	132173.281246674	132175.070127007	1.788880332984260	999.3949976	-0.061
P4	86000	87000	132175.070372435	132176.774762448	1.704390013008380	952.1927329	-4.781

FIFO_3							
process	理論start	理論end	dmesg start	dmesg end	delta	actually run unit	error(%)
P1	0	8000	132334.406367978	132347.353215493	12.946847515006100	7233.024145	-9.587
P2	8000	13000	132347.353712507	132356.287021218	8.933308711013520	4990.77768	-0.184
P3	13000	16000	132356.287305674	132361.728712060	5.441406386002200	3039.954223	1.332
P4	16000	17000	132361.729266334	132363.520953831	1.791687496996020	1000.963278	0.096
P5	17000	18000	132363.521232777	132365.235249802	1.714017024991340	957.5710622	-4.243
P6	18000	19000	132365.235562007	132366.999610799	1.764048791985260	985.5223437	-1.448
P7	19000	23000	132366.999900362	132373.840088259	6.840187897003490	3821.412445	-4.465

III. SJF

SJF_1							
process	理論start	理論end	dmesg start	dmesg end	delta	actually run unit	error(%)
P1	7000	14000	133212.315097831	133224.853836289	12.538738458009900	7005.02558	0.072
P2	0	2000	133200.051826886	133203.668287074	3.616460188000930	2020.410284	1.021
P3	2000	3000	133203.668552674	133205.528076528	1.859523853985590	1038.861462	3.886
P4	3000	7000	133205.528327571	133212.314609324	6.786281753011280	3791.296663	-5.218

SJF_2							
process	理論start	理論end	dmesg start	dmesg end	delta	actually run unit	error(%)
P1	100	200	133262.665509816	133262.846800303	0.181290487002116	101.2816802	1.282
P2	400	4400	133263.211433271	133270.147343366	6.935910094995050	3874.889631	-3.128
P3	200	400	133262.847038266	133263.211195971	0.364157705014804	203.4442338	1.722
P4	4400	8400	133270.147820853	133277.150492015	7.002671162015760	3912.187082	-2.195
P5	8400	15400	133277.150961542	133289.530977246	12.380015703995000	6916.35183	-1.195

SJF_3							
process	理論start	理論end	dmesg start	dmesg end	delta	actually run unit	error(%)
P1	100	3100	133318.821252645	133324.292236022	5.470983377017550	3056.478021	1.883
P2	11120	16120	133338.557573675	133347.478496858	8.920923183002740	4983.858249	-0.323
P3	16120	23120	133347.478995777	133360.085534864	12.606539086991700	7042.903803	0.613
P4	3100	3110	133324.292701130	133324.308312377	0.015611246984918	8.721546017	-12.785
P5	3110	3120	133324.308546309	133324.325562189	0.017015880002873	9.506273305	-4.937
P6	3120	7120	133324.325800097	133331.406373585	7.080573487997750	3955.708827	-1.107
P7	7120	11120	133331.406618073	133338.557069018	7.150450944987820	3994.747314	-0.131
P8	23120	32120	133360.085795914	133376.402735866	16.316939951997500	9115.795989	1.287

IV. PSJF

V. RR

VI. 可以發現以上的結果基本上都和實際情形很相近，誤差都很小。且排程順序也確實都和實際排程相同，畢竟我有保證時脈相同。

VII. 在極少數的狀況下會出現 soft block 的狀況，好像是 CPU 被鎖住了，不太確定為啥，不過不影響結果故無深究。

4. 解釋造成差異的原因
   A. 我認為我是在虛擬機上跑會造成一定的誤差，因為虛擬機對外部真正電腦來說，其實是在 user mode 執行的，就算用 sched_setscheduler，將 process 的 priority 設定 Real time 的最高 priority，也不一定真的整個 CPU3 都給他用了。
   B. 由於我有稍微使用 read, write，來做 process 間溝通，確保時脈，可能也會造成些微差異。
   C. 系統當下實際在做啥，我並不確定，或多或少也會造成差異。