B07902106　資工二　夏寧

2020 OS Project 1 - Process Scheduling

Design

►scheduler :

- 從*stdin*讀取*scheduling policy*和*N*個*process*的資訊
- 若為*PSJF*或*SJF* => 將*process*依照*execution time -> ready time*由小到大排列
  若為*FIFO*或*RR* => 將*process*依照*ready time*由小到大排列
- 把*process*分配到*scheduler*專用的*CPU*
- 調高*scheduler*的*priority*
- 初始化計時用的*clock*、正在執行的*process*、執行完畢的*process*數量、*mask*值等
- 跑*while*迴圈，每圈都是經過一個*unit*的一次*routine*

► scheduler每經過一個unit的routine :

- 檢查是否有*process*執行完畢，若有，則：
  - 用*wait*獲取該*process*的*exit status*
  - 檢查是否所有的*process*皆執行完畢，若是則*schedule*結束
- 找出所有在這一秒*ready*的*process*，把它們的*mask*值設為*1(*表已經*ready*好但尚未被*fork*出*)*
  - 若*policy*為*RR*，則將*ready*的*process*丟入*queue*中
- 挑出下一個要*execute(*佔用*CPU)*的*process (*若*next process*的*mask = 1*，這時才真正*fork*出該*process)*
- 若有成功選到下一個要執行的*process*且其不等於現在正在執行的*process*，則需要*context switch*：
  - 降低現在正在執行的*process*的*priority (SHCED_IDLE)*
  - 調高下個欲執行的*process*的*priority (SCHED_OTHER)*
  - 現在正在執行的*process* <= 下個欲執行的*process*
  - 記錄*context switch*的時間點
- *call function*經過*a unit of time*
- 若現在有正在執行的*process*，則將它的*execution time*減一
- 記錄時間的*clock*加一

► 選出下一個要執行的process的方法：

- First in, first out (FIFO)
  - policy：ready time愈早的愈先執行，且執行中不能被打斷
  - 作法：
    - 若*CPU*正在被某佔用，則讓那個*process*繼續使用*CPU(*因為是*non-preemptive)*
    - 若*CPU*閒置中，則找*ready time*最小的*process i(*也就是第*finished_proc_num*個*process*，*finished_proc_num*為已執行完畢的*process*的數量*)*：
      - 若*i*的*ready time*大於現在時間，則讓*CPU*繼續閒置
      - 若*i*的*ready time*小於等於現在時間，則將*i*選做下一個執行的*process*
- Round-Robin (RR)
  - policy：每個process一次最多只能跑一個time quantum = t 秒，t 秒到了若還沒跑完也必須讓出CPU
  - 作法：
    - 若*CPU*正在被*process i*佔用，則檢查是否已經跑一個*time quantum*了
      - 若已跑*t*秒了，則從*queue*中取出一個*process*來跑（若*queue*是空的則讓*CPU*閒置）
        - 如果*i*還沒執行完，則把他丟入*queue*中
      - 若還沒跑足*t*秒，則讓*i*繼續使用*CPU*
    - 若*CPU*閒置中，則從*queue*中取出一個*process*來跑（若*queue*是空的則讓*CPU*繼續閒置）
- Shortest job first (SJF)
  - Policy：execution time愈短的愈先執行，且執行中不能被打斷
  - 作法：
    - 若*CPU*正在被某佔用，則讓那個*process*繼續使用*CPU(*因為是*non-preemptive)*
    - 若*CPU*閒置中，則*linear*地找*ready time*小於等於現在時間，且*execution time*最小的*process*
      - 若所有尚未執行完畢的*process*的*ready time*都大於現在時間，則讓*CPU*繼續閒置

- Preemptive shortest job first (PSJF)
  - Policy：execution time愈短的愈先執行，且執行中可以被打斷
  - 作法：
    - *linear*地找*ready time*小於等於現在時間，且*execution time*最小的*process*
      - 若所有尚未執行完畢的*process*的*ready time*都大於現在時間，則讓*CPU*閒置

► process :
- *fork()*
  - *child process*：
    - 把*process name*、*process id*印到*stdout*
    - *call*自己寫的*system call(SYS_GET_SYSTEM_TIME(), syscall 333*號*)*得到開始的系統時間
    - 跑*execution time*個*unite of time*
    - *call*自己寫的*system call(SYS_GET_SYSTEM_TIME(), syscall 333*號*)*得到結束的系統時間
    - *call*自己寫的*system call(SYS_PRINTK(), syscall 334*號*)*把*process id*、開始/結束時間等資訊寫到*kernel*內部
  - *parent process*：
    - 降低*child process*的*priority (SHCED_IDLE)*
    - 把*child process*分配到*child process*專用的*CPU*

Kernel Version
Linux 4.14.25 x86_64

Results and Analysis
► 實際執行時間與理論執行時間的比較：
- 由TIME_MEASUREMENT執行出來的結果（在TIME_MEASUREMENT_dmesg.txt中）可算出500個unit大約是 0.762594（1個unit大約是 0.001525）。
- 以下turnaround time的算法皆為process end time (process結束的時間) - proves start time (process第一次搶到CPU的時間)
- FIFO：
  - FIFO_1：

	理論值	實際值	誤差
P1	0.762594	0.747072	0.015522
P2	0.762594	0.744725	0.017869
P3	0.762594	0.743622	0.018972
P4	0.762594	0.741817	0.020777
P5	0.762594	0.751883	0.010711
平均			0.016770

- FIFO_2：

	理論值	實際值	誤差
P1	122.015040	115.865890	6.149150
P2	7.625940	7.337958	0.287982
P3	1.525188	1.484848	0.040340
P4	1.525188	1.478033	0.047155
平均			1.631157

- FIFO_3：

	理論值	實際值	誤差
P1	12.201504	12.375128	0.173624
P2	7.625940	7.374987	0.250953
P3	4.575564	4.400425	0.175139
P4	1.525188	1.479645	0.045543
P5	1.525188	1.476919	0.048269
P6	1.525188	1.486475	0.038713
P7	6.100752	5.867962	0.232790
平均			0.137862

- FIFO_4：

	理論值	實際值	誤差
P1	3.050376	3.132250	0.081874
P2	0.762594	0.789004	0.026410
P3	0.305038	0.312370	0.007332
P4	0.762594	0.778797	0.016203
平均			0.032955

- FIFO_5：

	理論值	實際值	誤差
P1	12.201504	12.124527	0.076977
P2	7.625940	7.572979	0.052961
P3	4.575564	4.396615	0.178949
P4	1.525188	1.472867	0.052321
P5	1.525188	1.480333	0.044855
P6	1.525188	1.487527	0.037661
P7	6.100752	5.879737	0.221015
平均			0.094963

- RR：
  - RR_1：

	理論值	實際值	誤差
P1	0.762594	0.752759	0.009835
P2	0.762594	0.813654	0.051060
P3	0.762594	0.854601	0.092007
P4	0.762594	0.822510	0.059916
P5	0.762594	0.796302	0.033708
平均			0.049305

  - RR_2：

	理論值	實際值	誤差
P1	11.438910	11.690008	0.251098
P2	12.964098	13.221030	0.256932
平均			0.254015

  - RR_3：

	理論值	實際值	誤差
P1	28.215978	28.030083	0.185895
P2	26.690790	26.422815	0.267975
P3	21.352632	21.132881	0.219751
P4	38.129700	37.698338	0.431362
P5	35.841918	35.303052	0.538866
P6	30.503760	29.976455	0.527305
平均			0.361859

  - RR_4：

	理論值	實際值	誤差
P1	35.079324	34.590665	0.488659
P2	29.741166	29.416698	0.324468
P3	20.590038	20.615212	0.025174
P4	6.100752	6.298456	0.197704
P5	6.100752	6.263650	0.162898
P6	6.100752	6.268242	0.167490
P7	22.877820	22.506708	0.371112
平均			0.248215

- RR_5：

	理論值	實際值	誤差
P1	35.079324	35.174609	0.095285
P2	29.741166	29.886589	0.145423
P3	20.590038	20.820522	0.230484
P4	6.100752	6.012163	0.088589
P5	6.100752	6.095803	0.004949
P6	6.100752	6.186678	0.085926
P7	22.877820	23.077850	0.200030
平均			0.121527

- SJF：
  - SJF_1：

	理論值	實際值	誤差
P1	10.676316	11.461847	0.785531
P2	3.050376	3.236787	0.186411
P3	1.525188	1.651967	0.126779
P4	6.100752	6.264954	0.164202
平均			0.315731

  - SJF_2：

	理論值	實際值	誤差
P1	0.152519	0.163005	0.010486
P2	6.100752	6.384066	0.283314
P3	0.305038	0.336742	0.031704
P4	6.100752	6.468624	0.367872
P5	10.676316	11.578671	0.902355
平均			0.319146

  - SJF_3：

	理論值	實際值	誤差
P1	4.575564	4.968103	0.392539
P2	7.625940	8.117460	0.491520
P3	10.676316	11.351189	0.674873
P4	0.015252	0.018618	0.003366
P5	0.015252	0.020356	0.005104
P6	6.100752	6.593910	0.493158
P7	6.100752	6.487838	0.387086
P8	13.726692	14.324006	0.597314
平均			0.380620

- SJF_4：

	理論值	實際值	誤差
P1	4.575564	4.841443	0.265879
P2	1.525188	1.597051	0.071863
P3	6.100752	6.376013	0.275261
P4	3.050376	3.289382	0.239006
P5	1.525188	1.658868	0.133680
平均			0.197138

- SJF_5：

	理論值	實際值	誤差
P1	3.050376	3.265802	0.215426
P2	0.762594	0.842206	0.079612
P3	0.762594	0.798305	0.035711
P4	0.762594	0.820481	0.057887
平均			0.097159

- PSJF：
  - PSJF_1：

	理論值	實際值	誤差
P1	38.129700	40.098602	1.968902
P2	22.877820	24.588376	1.710556
P3	12.201504	13.327085	1.125581
P4	4.575564	5.047302	0.471738
平均			1.319194

  - PSJF_2：

	理論值	實際值	誤差
P1	6.100752	6.549076	0.448324
P2	1.525188	1.686619	0.161431
P3	10.676316	11.403962	0.727646
P4	3.050376	3.161606	0.111230
P5	1.525188	1.590034	0.064846
平均			0.302695

- PSJF_3：

	理論值	實際值	誤差
P1	5.338158	5.788387	0.450229
P2	0.762594	0.813869	0.051275
P3	0.762594	0.859355	0.096761
P4	0.762594	0.866658	0.104064
平均			0.175582

- PSJF_4：

	理論值	實際值	誤差
P1	10.676316	10.895389	0.219073
P2	4.575564	4.996548	0.420984
P3	1.525188	1.657905	0.132717
P4	6.100752	6.360143	0.259391
平均			0.258041

- PSJF_5：

	理論值	實際值	誤差
P1	0.152519	0.175452	0.022933
P2	6.100752	6.482900	0.382148
P3	0.305038	0.341294	0.036256
P4	6.100752	6.448282	0.347530
P5	10.676316	11.357319	0.681003
平均			0.293974

►觀察：
- 普遍來說，理論turnaround time愈久的process誤差值會愈大。

► 造成理論與實際誤差的原因：
- scheduler在設置priority、找下一個要執行的process、處理執行完畢的process等等動作都會耗費時間
- 實際可能存在除了scheduler跟被scheduler fork出來的process之外的其他process，如果context switch到那些process，就會造成實際花費的時間大於理論耗時。
- scheduler和child process的算過去多少unit of time的clock可能會有些許不一致，導致誤差（例如：若child process的clock略慢於scheduler的clock，那scheduler在wait的時候就需要花更多時間等待child process結束，造成實際耗時的增加）