OS Project 1

1. 設計

syscall.c

實作printk, getnstimeofday的接口。

process.c

process_block

block process的方式為將該process的schedule policy設定為`SCHED_IDLE`，以降低process的priority的方式來讓它盡量不被CPU執行。

process_wakeup

wake up process的方式是將該process的schedule policy設定為`SCHED_FIFO`，sched_priority為99，以讓process優先被CPU執行。

創建child process

child process在被創建後會把自己放到CHILD_CPU上，同時parent process會呼叫`process_block`來降低child process的priority。child process會持續檢查自己的priority是不是0並等待parent process調高自己的priority (wake up)。

當parent process認為輪到該child process執行時，parent process就會呼叫`process_wakeup`提高child process的priority，之後child process就會用`getnstimeofday`取得當前的時間，並輸出自己的名字與pid。我的start time紀錄的是process第一次丟上cpu執行的時間，而output則是在它第一次丟上cpu執行時印的。

當child process跑完時，它會用`getnstimeofday`取得當前的時間，並將pid, start time, finish time透過`printk`印到dmesg中。

scheduler.c

next_process

定義"能跑的process"為process的execution time > 0且current time >= ready time。按照policy的不同與當前正在執行的process的不同，會返回下一個應該要被執行的child process

- FIFO

  按照current time與ready time先後決定現在該誰跑

- RR

  如果目前的process已經執行了大於等於500個UNIT_TIME，就會傳回下一個能跑的process

- SJF

  若process還沒跑完，則返回正在執行的process，反之則從能跑的process中決定最短的execution time的process

- PSJF

每次都從能跑的process中決定最短的execution time的process

scheduling

在開始schedule前，parent process會將自己assign到PARENT_CPU，之後會對每個process按照ready time由小到大做排序。之後parent process便會開始模擬時間的進行，以一個UNIT_TIME為一單位，當child process的ready time來臨時，parent process就會create它並block它。接下來parent process會根據scheduling policy決定目前該跑的child process是誰，並block上個在跑的child process，然後wake up現在該跑的child process，並把該child process的execution time減1，代表它執行了一個UNIT_TIME。在執行過程中，如果parent process發現該child process的execution time變成0了，代表它應該要跑完了，所以parent process就會wait到它結束，以此來修正誤差。

main.c

讀取input並呼叫`scheduling`。

2. 核心版本

Linux 4.14.25

3. 比較實際結果與理論結果

手算出理論結果，再跟實際結果比較。

經測試一個unit time = 0.001688秒，以下的時間皆以unit time為單位，我用$(\text{finish time} - \text{start time})/0.001668$得到在這個執行時間下大概是多少unit time，以此來判斷誤差大小。結果如下:

TIME_MEASUREMENT.txt

process name	theoretical time	execution time	error	error percentage
P0	500	489.90	-10.10	-2.02%
P1	500	490.27	-9.73	-1.95%
P2	500	485.59	-14.41	-2.88%
P3	500	513.93	13.93	2.79%
P4	500	494.31	-5.69	-1.14%
P5	500	496.71	-3.29	-0.66%
P6	500	490.00	-10.00	-2.00%
P7	500	497.82	-2.18	-0.44%
P8	500	550.89	50.89	10.18%
P9	500	490.59	-9.41	-1.88%

FIFO_1.txt

process name	theoretical time	execution time	error	error percentage
P1	500	494.06	-5.94	-1.19%
P2	500	484.28	-15.72	-3.14%
P3	500	496.73	-3.27	-0.65%
P4	500	504.84	4.84	0.97%
P5	500	487.50	-12.50	-2.50%

PSJF_2.txt

process name	theoretical time	execution time	error	error percentage
P2	1000	999.13	-0.87	-0.09%
P1	4000	4032.34	32.34	0.81%
P4	2000	1964.80	-35.20	-1.76%
P5	1000	987.85	-12.15	-1.21%
P3	7000	6904.96	-95.04	-1.36%

RR_3.txt

process name	theoretical time	execution time	error	error percentage
P3	14000	13583.36	-416.64	-2.98%
P1	19000	18432.86	-567.14	-2.98%
P2	18000	17450.01	-549.99	-3.06%
P6	21000	20363.66	-636.34	-3.03%
P5	23500	22789.78	-710.22	-3.02%
P4	25000	24262.41	-737.59	-2.95%

SJF_4.txt

process name	theoretical time	execution time	error	error percentage
P1	3000	2937.37	-62.63	-2.09%
P2	1000	980.00	-20.00	-2.00%
P3	4000	3905.41	-94.59	-2.36%
P5	1000	979.40	-20.60	-2.06%
P4	2000	1966.87	-33.13	-1.66%

基本上大部分的誤差都在±3%內。大部分的時候實際的都比理論上的來的快，推測是因為有些process有略為偷跑的情形。