OS Project 1

1. 設計

syscall.c

實作`printk`, `getnstimeofday`的system call的接口。

process.c

定義Process型別如下:

```
typedef struct Process {
    pid_t pid;
    char name[32];
    int ready_time, exec_time;
} Process;
```

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 assign到CHILD_CPU上，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和自己有沒有被assign到PARENT_CPU上，並等待parent process調高自己的priority (wake up)與assign自己到CHILD_CPU。

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。當目前沒有process執行時，傳回第一個能跑的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 child process並block它。此時child process仍然在PARENT_CPU上，所以child process會被困在自己的while迴圈內直到parent process將它assign到CHILD_CPU上為止。
- 接下來parent process會呼叫`next_process`決定目前該跑的child process是誰，並block上一個在跑的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、宣告process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。

我先測出一個unit time大概是多少秒，再用$(\text{finish time} - \text{start time})/0.001625$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。以下的時間皆以unit time為單位:

mean absolute error percentage = 0.015503 = 1.5503%

以下舉幾個範例討論:

TIME_MEASUREMENT.txt

process name	theoretical time	execution time	error	error percentage
P0	500	509.36	9.36	1.87%
P1	500	499.40	-0.60	-0.12%
P2	500	498.80	-1.20	-0.24%
P3	500	494.42	-5.58	-1.12%
P4	500	495.95	-4.05	-0.81%
P5	500	497.22	-2.78	-0.56%
P6	500	490.95	-9.05	-1.81%
P7	500	508.64	8.64	1.73%
P8	500	505.48	5.48	1.10%
P9	500	499.78	-0.22	-0.04%

FIFO_1.txt

process name	theoretical time	execution time	error	error percentage
P1	500	506.30	6.30	1.26%
P2	500	494.41	-5.59	-1.12%
P3	500	492.09	-7.91	-1.58%
P4	500	503.19	3.19	0.64%
P5	500	497.22	-2.78	-0.56%

PSJF_2.txt

process name	theoretical time	execution time	error	error percentage
P2	1000	977.67	-22.33	-2.23%
P1	4000	3935.18	-64.82	-1.62%
P4	2000	1952.56	-47.44	-2.37%
P5	1000	983.21	-16.79	-1.68%
P3	7000	6851.00	-149.00	-2.13%

RR_3.txt

process name	theoretical time	execution time	error	error percentage
P3	14000	14136.47	136.47	0.97%
P1	19000	19181.11	181.11	0.95%
P2	18000	18195.28	195.28	1.08%
P6	21000	21157.63	157.63	0.75%
P5	23500	23665.51	165.51	0.70%
P4	25000	25098.39	98.39	0.39%

SJF_4.txt

process name	theoretical time	execution time	error	error percentage
P1	3000	3008.16	8.16	0.27%
P2	1000	1009.83	9.83	0.98%
P3	4000	4004.01	4.01	0.10%
P5	1000	1017.43	17.43	1.74%
P4	2000	2012.32	12.32	0.62%

基本上大部分的誤差都在$\pm 3\%$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。