OS Project 1

B07902072 資工二 陳光裕

設計-程式

共五個檔案:main.c my_process.c my_process.h my_scheduler.c my_scheduler.h

main.c

main.c主要是負責讀入資料，並且儲存在struct process中，最後呼叫scheduling進行排程。

my_process.c & my_process.h

process.h中定義了單位時間_unit_()，以及四個函式:

```
int process_assign_CPU(int pid, int core);
```
指定cpu，讓排程跟執行分開做。

```
int process_execute(struct process proc);
```
執行process，當某個process準備執行的時候，藉由此函式呼叫fork來放入不同的cpu執行，process也藉由此函式來得到pid。

```
int process_block(int pid);
int process_wakeup(int pid);
```
調整process的優先權。

my_scheduler.c & my_scheduler.h

負責排程。包含兩個函式:

```
int scheduling(struct process *proc, int proc_num, int name);
```
main最後會呼叫此函式，藉由while(1)來安排process。

```
int next_proc_id(struct process *proc, int proc_num, int name);
```
決定下個_unit_()時間要執行哪個process。

以及對FIFO、RR、SJF、PSJF分別給值，方便排程時判斷，減少strcmp所花的時間。

設計-排程方式

FIFO

由於先進先出，我們只要按照進來的順序，一個一個執行，就能完成排程。將所有process以進來的順序排序，從前面開始做，如果當前的running process尚未結束，就將next設為此process。

RR

設定每個process只能跑固定個unit(這裡是500個unit)，如果時間到，process尚未結束，就要換下一個process來執行，下一個process是由一個queue來決定的，每當新的process ready的時候，就把process放入queue中，並且每次都取出queue的頭當作next，如果時間到，process還沒結束，就把此process放回queue中，讓下一個process執行。

SJF

SJF是先把當前能執行的Process中，剩餘執行時間最少的Process拿出來執行。如果當前的running process尚未結束，就跟FIFO相同；如果結束了，就把當前ready的processes中選出剩餘執行時間最短的拿出來執行。

PSJF

PSJF是每當有新的process準備的時候，就檢查當前剩餘執行時間最少的是哪個process，因此每次決定next的時候都要確認是否有剛好開始ready的process，如果當前工作完成，就從ready的processes中選出剩餘執行時間最少的process來完成。

核心版本

linux-4.14.25

理論與實際的比較及原因

TIME_MEASUREMENT

```
yvecca@ubuntu:~/OS2020Project1$ dmesg | grep Project1
[10581.921519] [Project1] 3966 1588351662.951395805 1588351664.005474339
[10583.852865] [Project1] 3967 1588351664.960487521 1588351665.936823258
[10585.732549] [Project1] 3968 1588351666.860920375 1588351667.816508932
[10587.746248] [Project1] 3969 1588351668.828010030 1588351669.830209605
[10589.679556] [Project1] 3970 1588351670.785471900 1588351671.763519583
[10591.616166] [Project1] 3971 1588351672.701508249 1588351673.700130784
[10593.619870] [Project1] 3972 1588351674.734422161 1588351675.703836367
[10595.572581] [Project1] 3973 1588351676.675730747 1588351677.656550371
[10597.568513] [Project1] 3974 1588351678.603960052 1588351679.652483016
[10599.627944] [Project1] 3975 1588351680.720896415 1588351681.711914433
```

計算_unit_()，每個process跑500個unit平均花0.9954647333秒，也就是每unit大約花費0.0019909294666秒，大約是1500 1500秒。

利用以下測資來進行比較

FIFO_1.txt

```
FIFO
5
P1 0 500
P2 0 500
P3 0 500
P4 0 500
P5 0 500
```

這個測資與TIME_MEASUREMENT差不多，因此結果應該也會差不多

```
yvecca@ubuntu:~/OS2020Project1$ dmesg | grep Project1
[10624.738867] [Project1] 3991 1588351705.844327502 1588351706.822859021
[10625.811652] [Project1] 3992 1588351706.829218781 1588351707.895645626
[10626.807158] [Project1] 3993 1588351707.906420237 1588351708.891150730
[10627.809888] [Project1] 3994 1588351708.901774128 1588351709.893883271
[10628.778815] [Project1] 3995 1588351709.896267818 1588351710.862811307
```

理論花費時間為0.995、0.995、0.995、0.995、0.995，平均為0.995。
實際花費時間為0.978、1.066、0.984、0.992、0.966，平均為0.997。
實際與理論相差不多。

PSJF_2.txt

```
PSJF
5
P1 0 3000
P2 1000 1000
P3 2000 4000
P4 5000 2000
P5 7000 1000
```

```
yvecca@ubuntu:~/OS2020Project1$ dmesg | grep Project1
[10655.573327] [Project1] 4005 1588351735.692088787 1588351737.657251371
[10659.585692] [Project1] 4004 1588351733.679557485 1588351741.669590462
[10665.672732] [Project1] 4007 1588351743.727630630 1588351747.756593066
[10667.629295] [Project1] 4010 1588351747.758153795 1588351749.713142612
[10673.526157] [Project1] 4006 1588351741.773333605 1588351755.609967329
```

先執行P1 1000unit，換到P2 1000unit(結束)，換到P1 2000unit(結束)，P3 1000unit，P4 2000unit(結束)，P5 1000unit(結束)，最後回到P3 3000unit(結束)。

順序為P2、P1、P4、P5、P3。
理論花費時間為1.99、7.96、3.98、1.99、13.93。
實際花費時間為1.96、7.99、4.02、1.95、13.83。
實際與理論差不多。

RR_3.txt

```
RR
6
P1 1200 5000
P2 2400 4000
P3 3600 3000
P4 4800 7000
P5 5200 6000
P6 5800 5000
```

```
yvecca@ubuntu:~/OS2020Project1$ dmesg | grep Project1
[10728.494231] [Project1] 4019 1588351781.384929249 1588351810.577716659
[10729.406635] [Project1] 4017 1588351775.328310619 1588351811.490116411
[10730.359796] [Project1] 4018 1588351778.286784847 1588351812.443272136
[10748.743017] [Project1] 4022 1588351787.825270365 1588351830.826389073
[10752.855071] [Project1] 4021 1588351786.821483047 1588351834.938424419
[10754.648848] [Project1] 4020 1588351785.749854528 1588351836.732192000
```

Queue的順序為
P1->P1->P1P2->P1P2P3->P1P2P3P4->P1P2P3P4P5->P1P2P3P4P5P6->P1P2P3P4P5P6->P1P2P3P4P5P6->P1P2P4P5P6->P4P5P6->P4P5P6->P4P5P6->P4P5P6->P4P5P6->P4P5->P4

順序為P3、P1、P2、P6、P5、P4。
理論花費時間為24.875、34.825、33.83、35.82、42.785、49.75
實際花費時間為29.192、36.151、34.15、43.00、48.116、50.98
實際與理論有點小誤差，應該是因為在process之間switch的次數比其他排程方式來的多，時間差累積起來，就造成了誤差。

SJF_4.txt

```
SJF
5
P1 0 3000
P2 1000 1000
P3 2000 4000
P4 5000 2000
P5 7000 1000
```

```
yvecca@ubuntu:~/OS2020Project1$ dmesg | grep Project1
[10781.199491] [Project1] 4032 1588351857.190677695 1588351863.282695510
[10783.147152] [Project1] 4033 1588351863.284352029 1588351865.230346858
[10791.194900] [Project1] 4034 1588351865.231745002 1588351873.278053278
[10793.143972] [Project1] 4038 1588351873.280440768 1588351875.227116136
[10797.049504] [Project1] 4037 1588351875.253308270 1588351879.132629379
```

先執行P1 3000，當時間為1000時P2 wait，時間為2000時P3 wait，P1結束後換執行時間較小的P2 1000，然後執行P3 3000，當時間為5000的時候P4 wait，當時間為7000的時候P3結束，選較小的P5 1000，最後P4 2000。

順序為P1、P2、P3、P5、P4。
理論花費時間為5.97、1.99、7.96、1.99、3.97。
實際花費時間為6.09、1.94、8.04、1.94、3.87。
實際與理論相差不多。