

作業系統 2020 Project1

R08922158 黃聲凱

核心版本：

Kernel 版本：Linux4.14.25

Platform：Ubuntu 16.04

設計概念：

使用兩顆 cpu，將行程分配給特定的 cpu，以此避免 scheduler 和 forked processes 之間的搶佔。實現四個排程策略，透過 function 來調整優先級，使行程在一顆 cpu 上依排程策略執行，紀錄行程的 start time 和 end time，再與理論時間做對比。

在 while 迴圈中，1.檢查是否有正在執行的行程已經完成執行。如果所有行程都已經執行完，則迴圈中斷。2.檢查是否有行程準備就緒。3.確定在下一個時間單元中要執行的行程。4.檢查是否需要結束正在執行的行程並且啟動下一個行程。5.定義 a unit time，執行 unit_time()。

main.c

進行讀檔：scheduling policy, process name, ready time 和 execution time；
根據排程策略開始進行排程。

priority.c

透過兩個 function 來提高/降低(raise_priority(), reduce_priority())排程的優先權。

process.c

提供 function 來控制 process：

unit_time()用來定義一個 unit 的執行時間。

assign_proc_to_cpu()用來限制 process 在特定 cpu 上執行。

exec_process()用來 fork 行程：

首先 fork 子行程，使用 run unit_time()根據行程的執行時間紀錄 start time；紀錄 end time，使用系統呼叫將 start time 和 end time 打印到 dmesg 中。最後打印行程 name 和 pid。

其次，在父行程中，透過 reduce_priority()模擬子行程就緒狀態，以此來降低子行程的優先級。

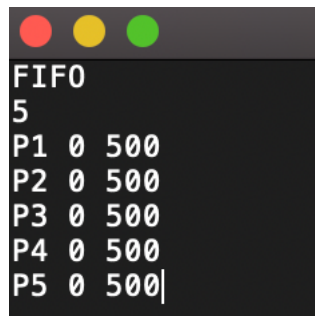
schedule.c :

被主程式呼叫的 function，從這裡再去執行選擇的排程策略：
FIFO 排程實現、RR 排程實現、SJF 排程實現、PSJF 排程實現

運行結果：

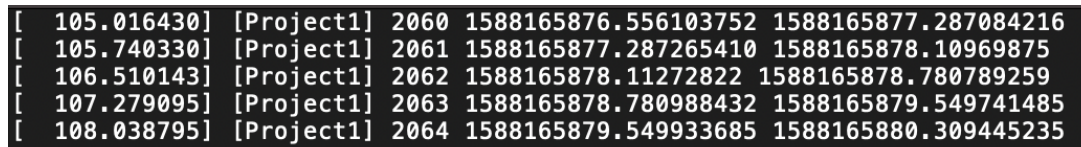
FIFO_1.txt

Input:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top. The text inside the terminal is as follows:

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

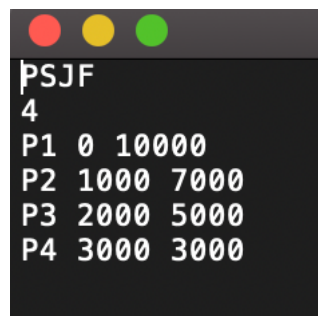
Output:

A terminal window with a dark background showing the output of the FIFO scheduling algorithm. The text is as follows:

```
[ 105.016430] [Project1] 2060 1588165876.556103752 1588165877.287084216
[ 105.740330] [Project1] 2061 1588165877.287265410 1588165878.10969875
[ 106.510143] [Project1] 2062 1588165878.11272822 1588165878.780789259
[ 107.279095] [Project1] 2063 1588165878.780988432 1588165879.549741485
[ 108.038795] [Project1] 2064 1588165879.549933685 1588165880.309445235
```

PSJF_1.txt

Input:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top. The text inside the terminal is as follows:

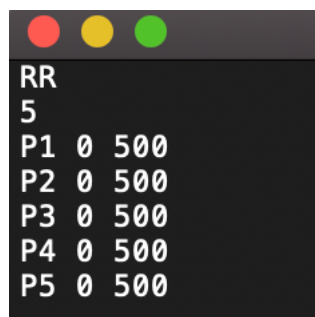
```
PSJF
4
P1 0 10000
P2 1000 7000
P3 2000 5000
P4 3000 3000
```

Output:

```
[ 1119.921145] [Project1] 2264 1588166889.42638688 1588166893.580455782
[ 1126.033085] [Project1] 2263 1588166887.462651934 1588166899.691800264
[ 1135.313515] [Project1] 2262 1588166885.873383990 1588166908.971334456
[ 1149.117921] [Project1] 2261 1588166884.350138605 1588166922.774387362
```

RR_1.txt

Input:



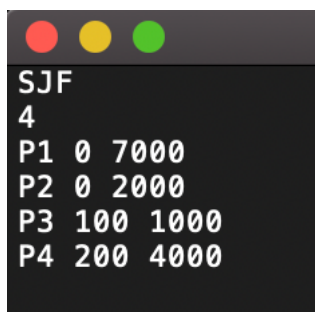
```
RR
5
P1 0 500
P2 0 500
P3 0 500
P4 0 500
P5 0 500
```

Output:

```
1854.507080] [Project1] 2418 1588167627.577523172 1588167628.300300766
1855.270919] [Project1] 2419 1588167628.300571550 1588167629.64350006
1856.087323] [Project1] 2420 1588167629.64546202 1588167629.880977891
1856.806460] [Project1] 2421 1588167629.881158413 1588167630.600307188
1857.530504] [Project1] 2422 1588167630.600703206 1588167631.324547136
```

SJF_1.txt

Input:



```
SJF
4
P1 0 7000
P2 0 2000
P3 100 1000
P4 200 4000
```

Output:

```
[ 2447.220951] [Project1] 2528 1588168217.676356046 1588168221.56102225  
[ 2448.793430] [Project1] 2529 1588168221.70251279 1588168222.628231286  
[ 2455.477155] [Project1] 2530 1588168222.628526936 1588168229.310545814  
[ 2466.864719] [Project1] 2527 1588168221.56334396 1588168240.695935594
```

理論與實際結果差異總結：

基本上實際值不管在 **start time** 和 **end time** 都是比理論值大，且越晚的 **start time** 和 **end time** 誤差會變大，這是因為排程的 **process** 在 **while** 迴圈中需要做別的事情：**ready process**、調整優先級、確定下一個執行的行程。所以一個 **while** 迴圈中並不是只有做一次 **unit time**。做別的事情時會浪費子行程所用的 **cpu** 的 **cpu** 時間，因此時間越久誤差將會越大。由於在虛擬機上運行代碼，因此存在不穩定性。我們選擇下一個行程的方式是反覆檢查每個行程並從就緒佇列中選擇一個行程，因此行程數可能會影響 **scheduling time**。**machine** 可能還有其他行程，因此我們創建的行程可能會受到 **context switch** 影響。即使運行相同數量的迴圈，不同的行程可能有不同的執行時間。